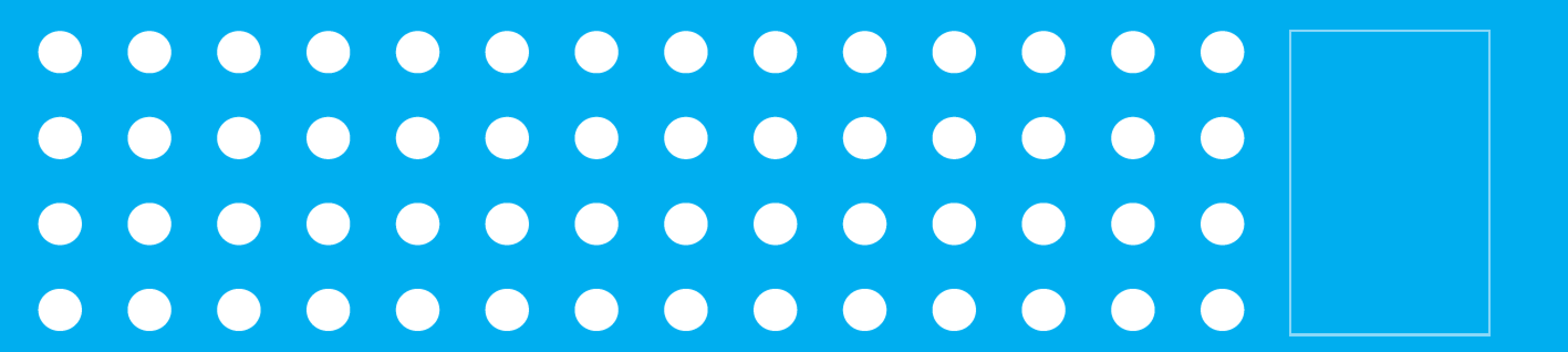
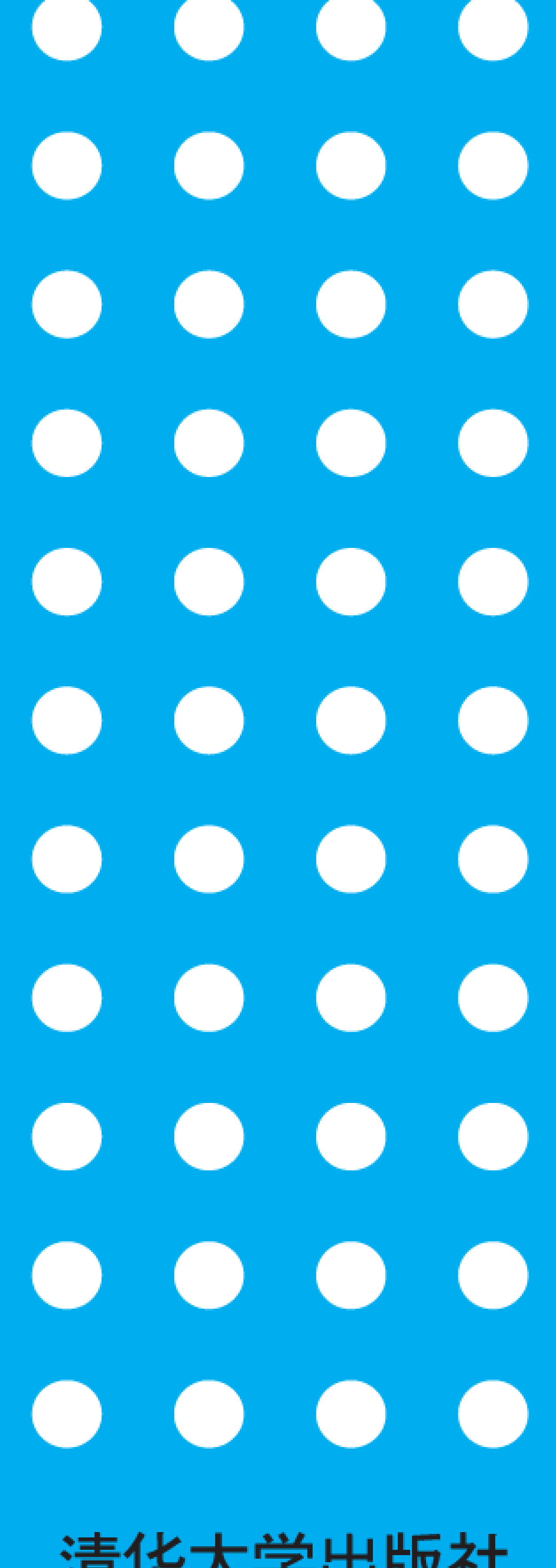


高职高专信息技术类专业项目驱动模式规划教材 / 丛书主编 赵有生

Web应用开发实训教程——JSP+Struts 2

孙佳帝 孙凌玲 刘洋 主编



清华大学出版社

高职高专信息技术类专业项目驱动模式规划教材

Web 应用开发实训教程——JSP＋Struts 2

孙佳帝　孙凌玲　刘　洋　主编

清华大学出版社
北　京

内 容 简 介

本书全面介绍使用 JSP、Servlet、Struts 2 框架技术等开发 Java EE 应用程序的方法，书中对 JSP、Servlet、Struts 2 的开发思想及技术要点进行了详细阐述。

本书共分两篇。第一篇为 JSP 与 Servlet，主要介绍搭建 Java EE 应用开发环境、JSP 基本语法、JSP 内置对象、JavaBean 与 JDBC、Servlet 及过滤器 Filter 等。第二篇为 Struts 2，主要介绍 Struts 2 核心原理、Action 应用详解、类型转换器、输入校验、访问 Web 元素、结果类型、OGNL 表达式与常用标签、拦截器等。最后通过论坛管理系统介绍如何综合应用 Struts 2 框架技术。

本书注重编程思想与实际开发相结合，书中的每个知识点都配备了具有典型性和实用价值的应用开发实例，使读者不仅能够掌握相关技术，更能够活学活用、举一反三。

本书可作为高等职业学院软件技术专业专科及本科学生的教材，也可供与计算机相关专业的技术人员使用。

图书在版编目（CIP）数据

Web 应用开发实训教程：JSP＋Struts 2/孙佳帝，孙凌玲，刘洋主编.--北京：清华大学出版社，2014
高职高专信息技术类专业项目驱动模式规划教材
ISBN 978-7-302-36446-7

Ⅰ. ①W…　Ⅱ. ①孙…②孙…③刘…　Ⅲ. ①JAVA 语言－网页制作工具－高等职业教育－教材　Ⅳ. ①TP312 ②TP393.092

中国版本图书馆 CIP 数据核字(2014)第 095810 号

责任编辑：孟毅新
封面设计：
责任校对：袁　芳
责任印制：

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦 A 座　**邮　　编**：100084
社 总 机：010-62770175　**邮　　购**：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质量反馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
课件下载：http://www.tup.com.cn，010-62795764
印 刷 者：
装 订 者：
经　　销：全国新华书店
开　　本：185mm×260mm　**印　　张**：23.5　**字　　数**：540 千字
版　　次：2014 年 8 月第 1 版　**印　　次**：2014 年 8 月第1 次印刷
印　　数：1～　000
定　　价：　.00 元

产品编号：048793-01

前言

伴随着 Internet 的飞速发展，软件开发已由传统的 C/S 模式逐渐转变到了现今流行的 B/S 模式。在基于 B/S 模式的 Web 开发工具中，Java 语言以其得天独厚的优势获得了广大开发人员的认可。随着 Java 的风起云涌，各种各样的框架接连出现，而 Struts 2 框架正是在这个时候悄然浮出水面。

本书编写的思想是“学科体系 + 工学结合”，也就是说，本书内容的组织是在学科体系结构基础之上，融入了工作过程导向的原理。所有内容按照学科体系、知识组成结构进行划分，从简单到复杂，由基础到深入。每个技术点都有足够的例子进行讲解，当技术点积累到一定程度的时候，就通过实践任务，以 step by step 的方式综合应用各个技术点。这样，既保证了知识体系结构的完整性，也对相关的技术点按照工作过程导向的原理进行了综合的实践，使学生能够系统全面地理解知识，并掌握如何在实践中应用相关的技术。

本书共分两篇，第一篇为 JSP 与 Servlet，第二篇为 Struts 2 框架。

第一篇共 8 章，详细介绍 JSP、JavaBean、Servlet、Filter、Listener 等 Java Web 常用的基本技术，并根据需要加入实践任务和拓展任务。通过实践任务的分析和实施，告诉读者如何在实践中应用相关的技术，提高读者的实践操作能力和综合应用能力。

第二篇共 11 章，详细介绍 Struts 2 框架的工作原理、控制器 Action、输入校验、类型转换器、OGNL 表达式与常用标签、拦截器等，并加入了大量的案例、实践任务及拓展任务，使读者由学会到会用。

本书聘请智翔集团的高级软件工程师弓建明参编。作为项目经理，他有着丰富的实践开发经验。他对本书的编写模式、项目设计思想、编码规范等方面给予指导，并根据企业常用的实际知识和技能，设计实现本书的实践任务，以任务引领技术点，拓展程序设计思路，提高学生的实践技能。

本书由孙佳帝、孙凌玲、刘洋主编，参编人员有李季、张立辉、于艳华、吴艳平、张静、王依楠、于薇、丁磊、闫雪、贺晶伟、乔丹。

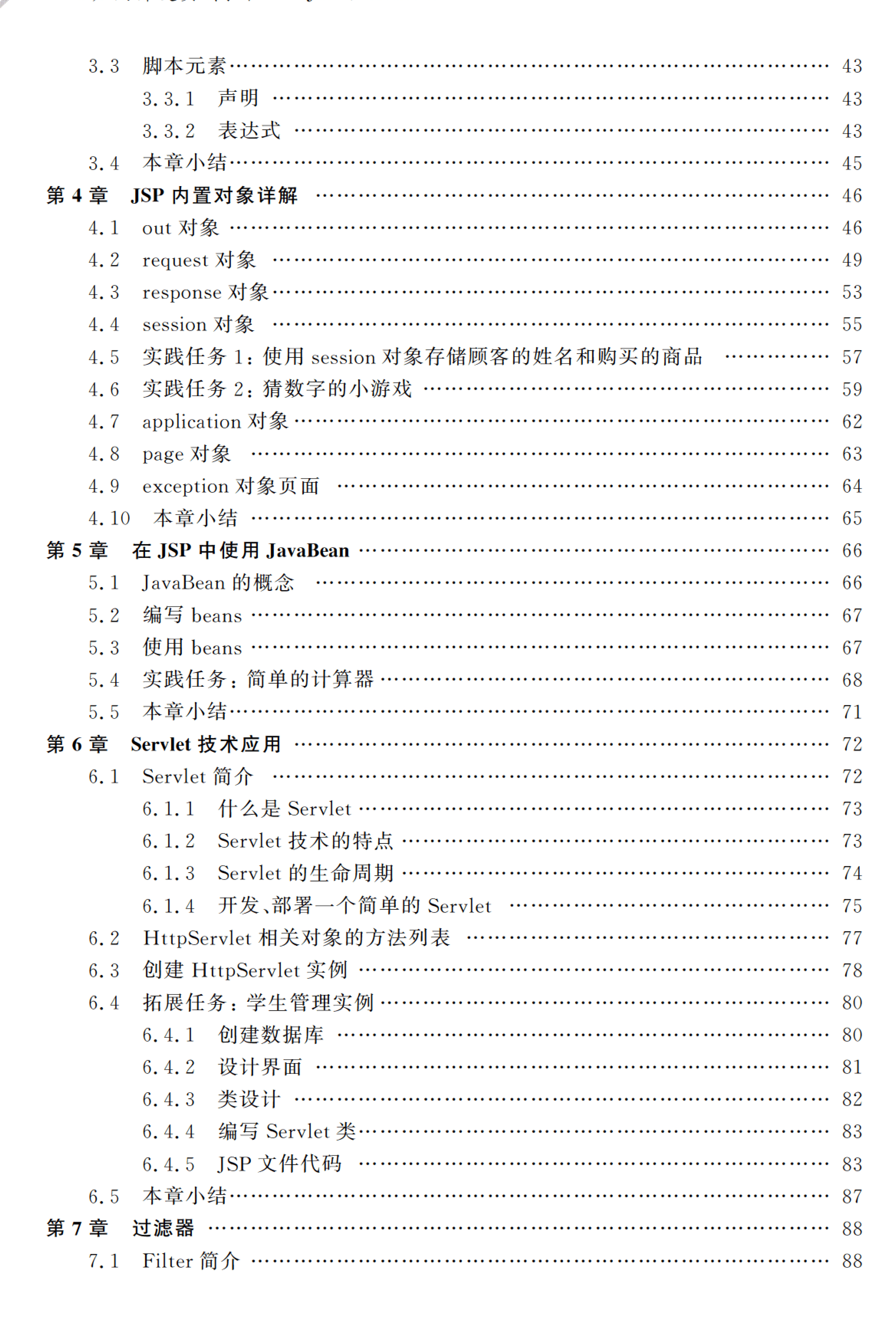
由于编者学识水平有限，虽竭智尽力，仍难免有不足之处，恳请广大读者批评、指正，以便今后修订完善。

作　者

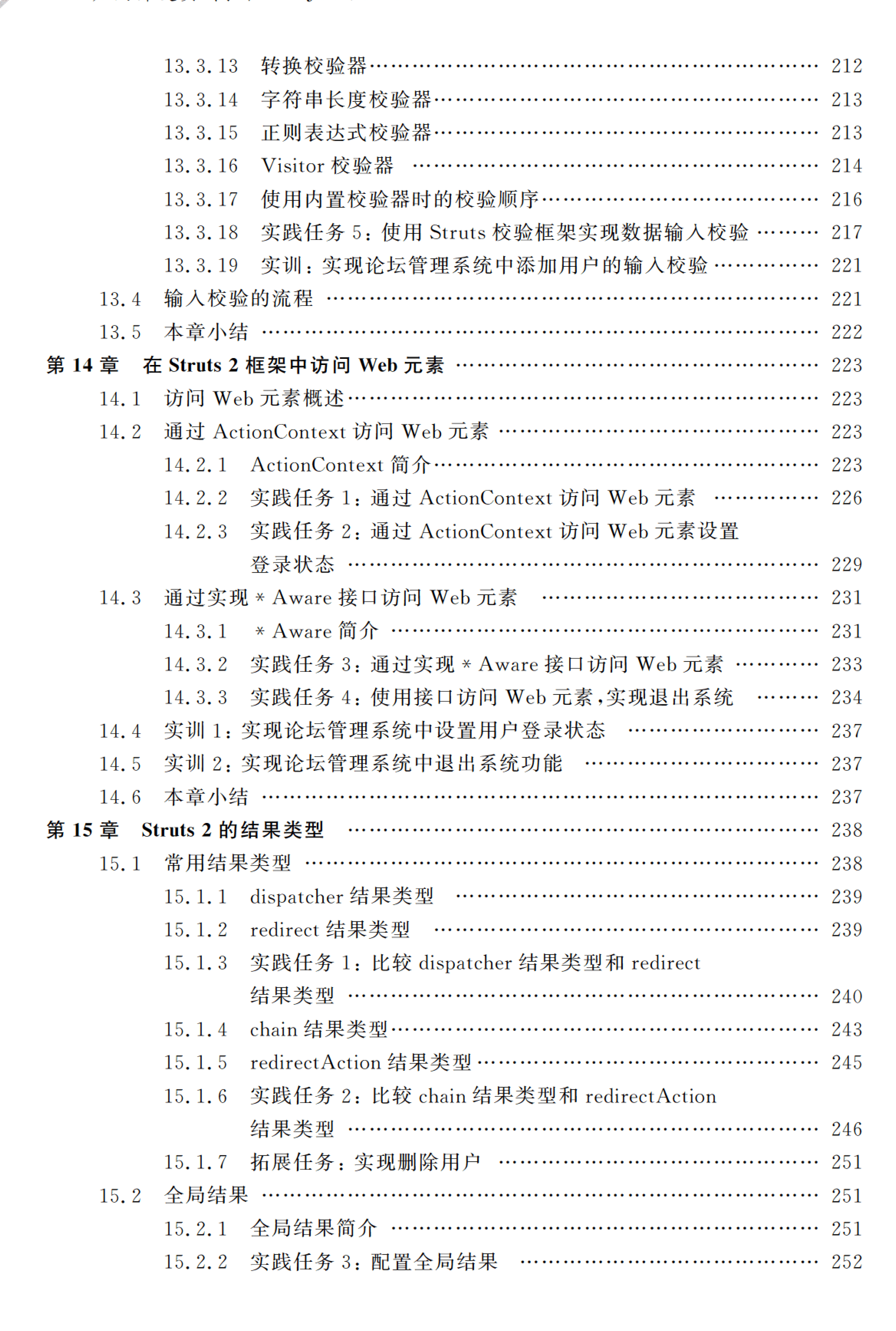
2014 年 6 月

目录

第一篇　JSP 与 Servlet



第二篇　Struts 2



第一篇

JSP 与 Servlet

第1章

JSP 技术简介

随着 Internet 技术的迅猛发展与 Java 语言的不断完善优化，越来越多的 Web 程序员义无反顾地踏上了 Java Web 应用开发之路。那么什么是 JSP？什么是 Servlet？它们与 Java 语言有什么关系呢？下面我们就开始 JSP 技术的学习之路。

本章要点：

- 常用的动态网页介绍
- Java、JSP、Servlet 的联系
- JSP 工作原理
- 为什么使用 JSP
- 开发第一个 JSP 页面

1.1 什么是 JSP

JSP 全称是 Java Server Pages，是由 Sun 公司倡导、许多公司参与，于 1999 年推出的一种动态网页技术标准。在传统的网页 HTML 文件(＊.htm，＊.html)中加入 Java 程序片段(Scriptlet)和 JSP 标记(Tag)，就构成了 JSP 网页(＊.jsp)。网页一般又称 HTML 文件，是一种可以在 WWW 上传输、能被浏览器认识和翻译成页面并显示出来的文件。目前网页根据生成方式，大致可以分为静态网页和动态网页两种。JSP 是基于 Java Servlet 以及整个 Java 体系的 Web 开发技术，利用这一技术可以建立安全、跨平台的先进动态网站。JSP 以 Java 技术为基础，又在许多方面做了改进，具有动态页面与静态页面分离、能够脱离硬件平台的束缚，以及编译后运行等优点。到目前为止，JSP 技术已经逐渐成为 Internet 上的主流开发技术。

需要强调的一点是：要想真正地掌握 JSP 技术，必须有较好的 Java 语言基础，以及 HTML 语言方面的知识。

下面是一个比较简单的 JSP 网页。

```
<%@ page contentType="text/html;charset=GB2312"%>
<%@ page import="java.util.*"%>
<HTML>
<BODY>
<P> 现在的时间是：
      <% Date date=new Date();%>
<BR>
```

```
<%=date%>
</BODY>
</HTML>
```

1.2 动态网页技术

这里说的动态网页，与网页上的各种动画、滚动字幕等视觉上的“动态效果”没有直接关系。动态网页可以是纯文字内容的，也可以是包含各种动画的内容，这些只是网页具体内容的表现形式。无论网页是否具有动态效果，采用动态网站技术生成的网页都称为动态网页。动态网页技术有以下几个特点。

(1) 交互性。即网页会根据用户的要求和选择而动态改变和响应，将浏览器作为客户端界面，这将是今后 WEB 发展的大势所趋。

(2) 自动更新。即无须手动更新 HTML 文档，便会自动生成新的页面，可以大大节省工作量。

(3) 因时因人而变。即当不同的时间、不同的人访问同一网址时会产生不同的页面。

目前，最常用的三种动态网页技术有 ASP(Active Server Pages)、JSP(Java Server Pages)、PHP(PHP: Hypertext Preprocessor)。下面，我们分别从技术和商业的角度来简单介绍这些动态网页技术。

1. ASP

(1) 技术角度

① 简介：ASP 又被称为服务器端的 VBScript，所以采用我们非常熟悉的 VB 语法。

② 公司特征：由 Microsoft 公司推出。

③ 难易程度：由于采用 Basic 语法，只要熟悉 Basic 语言，有点 HTML 基础，要学习掌握 ASP 是很简单的事情。

④ 代码隐藏性：不需要编译，直接运行，所以代码可视。

⑤ 编写工具：任何文本编辑器都可以进行编辑。

⑥ 封装性：能很好地结合 MS 的 COM+技术，可以将比较复杂的事务处理工作封装在 COM+中，而且能非常好地进行调用。

⑦ 扩充性：由于 MS 本身的 ActiveX 技术具有无限可扩充性，所以能很好地结合其他语言编写的组件。

⑧ 数据库：采用 MS 的 ODBC 接口技术，所以，几乎所有的数据库都可以结合。

⑨ 平台性：遗憾的是，对于我们目前流行的两种最主流的网络操作系统(UNIX/Linux 和 Windows)，它却只能在 Windows 平台上受到很好的支持。

(2) 商业角度

① 平台成本：可以选择 Windows Server+IIS+ASP+MS SQL Server 方案，因为这些都是 ASP“母”公司的产品，所以，从性能方面考虑，对于 ASP 肯定是最佳的方案，但这些产品中，几乎都是要支付一定的费用，对于一个小型公司来说，建设一个 ASP 的网站是比较“贵”的。

② 人工成本：ASP 技术简单，ASP 人才几乎到处可见，所以，开发一个 ASP 网站、维护一个 ASP 网站，单从人工成本上来说，是不需要太多费用的。

2. JSP

（1）技术角度

① 简介：JSP 全名 Java Server Pages，采用 Java 语法，Java 体系的任何东西都是需要 JDK 支持的，同样，JSP 也离不开 JDK。

② 公司特征：由 Sun 公司推出。

③ 难易程度：由于采用 Java 语法，而 Java 具有高度的面向对象和灵活性，所以，比 Basic 稍微要难点。

④ 代码隐藏性：要编译成 Servlet，在服务器端运行，所以代码不可视。

⑤ 编写工具：任何文本编辑器都可以进行编辑，但是，却需要 JDK 先编译好。

⑥ 封装性：能很好地结合 JavaBean 技术，可以将复杂的事务处理工作封装在 Bean 中，而 JSP 能非常好地进行调用。

⑦ 数据库：采用 Java 结合数据库技术，即 JDBC 技术，也是一个统一的数据库接口技术。

⑧ 平台性：对于目前流行的两种最主流的网络操作系统（UNIX/Linux 和 NT/Windows 2000），都能得到很好的支持。

（2）商业角度

① 平台成本：有很多可以考虑的非常优秀的方案，但是，这些“非常优秀”的方案中，其中的价格也是非常昂贵的。当然，也可以采用一些不需要支付任何费用的方案，如 Linux＋Tomcat＋JSP＋MySQL 等。

② 人工成本：从目前来看，由于 JSP 的诞生没有多长时间，而且，要真正将 Java 体系掌握清楚，并不是短时间就能做到的，所以，目前 JSP 人才相对来说，应该要比 ASP 和 PHP 人才少得多，从商业角度分析，成本也要贵不少。

3. PHP

（1）技术角度

① 语法：PHP，采用类似 C 的语法。

② 难易程度：由于采用 C 语言语法，所以要学习掌握 PHP 也是比较简单的事情。

③ 代码隐藏性：不需要编译，直接运行，所以代码可视。但是，可以通过 Zend 的编译器将其代码加密处理，以隐藏源代码。

④ 编写工具：任何文本编辑器都可以进行编辑。

⑤ 封装性：能结合 MS 的 COM＋技术，也能结合 JavaBean，将某些复杂的事务处理工作封装在 COM＋ 和 JavaBean 中，但是，性能当然没有像 ASP 结合 COM＋和 JSP 结合 JavaBean 那么“原版”，也就是速度和性能肯定没有后者好。

⑥ 公司特征：PHP 是开源的，所以，你可以得到它的源代码，并可以重新编译，甚至加入自己的特征。

⑦ 数据库：对于目前流行的数据库，几乎都有支持。但是，它却不是像 ASP、JSP 那样有个统一的接口，对于每种数据库几乎都有一个不同的接口。

⑧ 平台性：对于目前流行的两种最主流的网络操作系统(UNIX/Linux 和 Windows)，它都可以得到很好的支持，而且根本不用修改任何代码。

(2) 商业角度

① 平台成本：可以考虑 Linux+Apache+PHP+MySQL 方案，因为这个方案是 PHP 的最佳选择。而且令人兴奋的是，这些都是开放源代码的产品。也就是说，可以不向任何人支付任何费用就可以做到的。

② 人工成本：从目前来看，由于 PHP 目前也是非常流行的，所以，PHP 的人才也非常多。而且，开发速度也是非常快，所以从这个方面考虑，成本也是不高的。

4. 前景分析

对于比较大型的网站，比如对事务处理和负载均衡要求比较高的站点，采用 JSP 和 ASP 的比较多。从成本上考虑，比较经济的站点采用 PHP 应该是最好的选择。

由于三种语言各有自己的长处，所以，都有相当的支持者，在今后相当一段时间内，都不会被对方所淘汰。

1.3 Java、Servlet 和 JSP 的联系

本节将要简单介绍 Java、Servlet 和 JSP 的发展及联系。

1. Java

Java 是一种可以撰写跨平台应用软件的面向对象的程序设计语言，是由 Sun Microsystems 公司于 1995 年 5 月推出的 Java 程序设计语言和 Java 平台(即 Java SE、Java EE、Java ME)的总称。Java 技术具有卓越的通用性、高效性、平台移植性和安全性，广泛应用于个人 PC、数据中心、游戏控制台、科学超级计算机、移动电话和互联网，同时拥有全球最大的开发者专业社群。在全球云计算和移动互联网的产业环境下，Java 更具备了显著优势和广阔前景。

与传统程序不同，Sun 公司在推出 Java 之际就将其作为一种开放的技术。全球数以万计的 Java 开发公司被要求所设计的 Java 软件必须相互兼容。“Java 语言靠群体的力量而非公司的力量”是 Sun 公司的口号之一，并获得了广大软件开发商的认同。这与微软公司所倡导的注重精英和封闭式的模式完全不同。Sun 公司对 Java 编程语言的解释是：Java 编程语言是个简单、面向对象、分布式、解释性、稳健、安全与系统无关、可移植、高性能、多线程和动态的语言。Java 平台是基于 Java 语言的平台，这样的平台目前非常流行。因此，微软公司推出了与之竞争的.NET 平台以及模仿 Java 的 C# 语言。

2. Servlet

Servlet 是一种服务器端的 Java 应用程序，具有独立于平台和协议的特性，可以生成动态的 Web 页面。它担当客户请求(Web 浏览器或其他 HTTP 客户程序)与服务器响应(HTTP 服务器上的数据库或应用程序)的中间层。Servlet 是位于 Web 服务器内部的服务器端的 Java 应用程序，与传统的、从命令行启动的 Java 应用程序不同，Servlet 由 Web 服务器进行加载，该 Web 服务器必须包含支持 Servlet 的 Java 虚拟机。

服务器上需要一些程序，常常是根据用户输入访问数据库的程序。这些通常是使用公共网关接口(Common Gateway Interface，CGI)应用程序完成的。

在通信量大的服务器上，Java Servlet 的优点在于，它们的执行速度更快于 CGI 程序。各个用户请求被激活成单个程序中的一个线程，而不必创建单独的进程，这意味着服务器端处理请求的系统开销将明显降低。

Java Servlet 是 Java 语言的一部分，提供了用于服务器编程的 API。用 Java Servlet 编写的 Java 程序，称为一个 Servlet。Servlet 通过 HTML 与客户交互信息。Servlet 的最大缺点是，不能有效地管理页面的逻辑部分和页面的输出部分，导致 Servlet 代码非常混乱，用 Servlet 来管理网站变成一件很困难的事情。为了克服 Servlet 的缺点，SUN 公司用 Java Servlet 作为基础，推出了 Java Server Pages，即 JSP。

3. JSP

JSP 是由 Sun Microsystems 公司倡导、许多公司参与一起建立的一种动态网页技术标准。JSP 提供了 Servlet 的几乎所有好处，当一个客户请求一个 JSP 页面时，JSP 引擎根据 JSP 页面生成一个 Java 文件，即一个 Servlet。用 JSP 支持 JavaBean 这一特点，可以有效地管理页面的逻辑部分和页面的输出部分。另外，JSP 也可以和 Servlet 有效地结合，分离页面的逻辑部分和页面的输出部分。JSP 技术有点类似 ASP 技术，它是在传统的网页 HTML 文件(*.htm、*.html)中插入 Java 程序段(Scriptlet)和 JSP 标记(Tag)，从而形成 JSP 文件(*.jsp)。用 JSP 开发的 Web 应用是跨平台的，既能在 Linux 下运行，也能在其他操作系统上运行。

JSP 技术具有如下特点。

(1) 一次编写，到处运行。除了系统之外，代码不用做任何更改。

(2) 系统的多平台支持。基本上可以在所有平台上的任意环境中开发，在任意环境中进行系统部署，在任意环境中扩展。相比 ASP/.NET 的局限性是显而易见的。

(3) 强大的可伸缩性。从只有一个小的 Jar 文件就可以运行 Servlet/JSP，到由多台服务器进行集群和负载均衡，Java 显示了巨大的生命力。

(4) 多样化和功能强大的开发工具支持。这一点与 ASP 很像，Java 已经有了许多非常优秀的开发工具，而且许多可以免费得到，并且其中许多已经可以顺利地运行于多种平台之下。

(5) 支持服务器端组件。Web 应用需要强大的服务器端组件来支持，开发人员需要利用其他工具设计实现复杂功能的组件供 Web 页面调用，以增强系统性能。JSP 可以使用成熟的 JavaBean 组件来实现复杂的商务功能。

1.4　JSP 工作原理

JSP 工作原理如图 1.1 所示。当服务器上的一个 JSP 页面被第一次请求执行时，服务器上的 JSP 引擎首先将 JSP 页面文件转译成一个 Java 文件，再将这个 Java 文件编译生成字节码文件，然后通过执行字节码文件响应客户的请求，而当这个 JSP 页面再次被请求执行时，JSP 引擎将直接执行这个字节码文件来响应客户，这也是 JSP 比 ASP 速度

快的一个原因。而 JSP 页面的首次执行往往由服务器管理者来执行。这个字节码文件的主要工作如下。

(1) 把 JSP 页面中普通的 HTML 标记符号(页面的静态部分)交给客户的浏览器负责显示。

(2) 执行"<%"和"%>"之间的 Java 程序段(JSP 页面中的动态部分),并把执行结果交给客户的浏览器显示。

(3) 当多个客户请求同一个 JSP 页面时,JSP 引擎为每个客户启动一个线程而不是启动一个进程,这些线程由 JSP 引擎服务器来管理,与传统的 CGI 为每个客户启动一个进程相比较,效率要高得多。

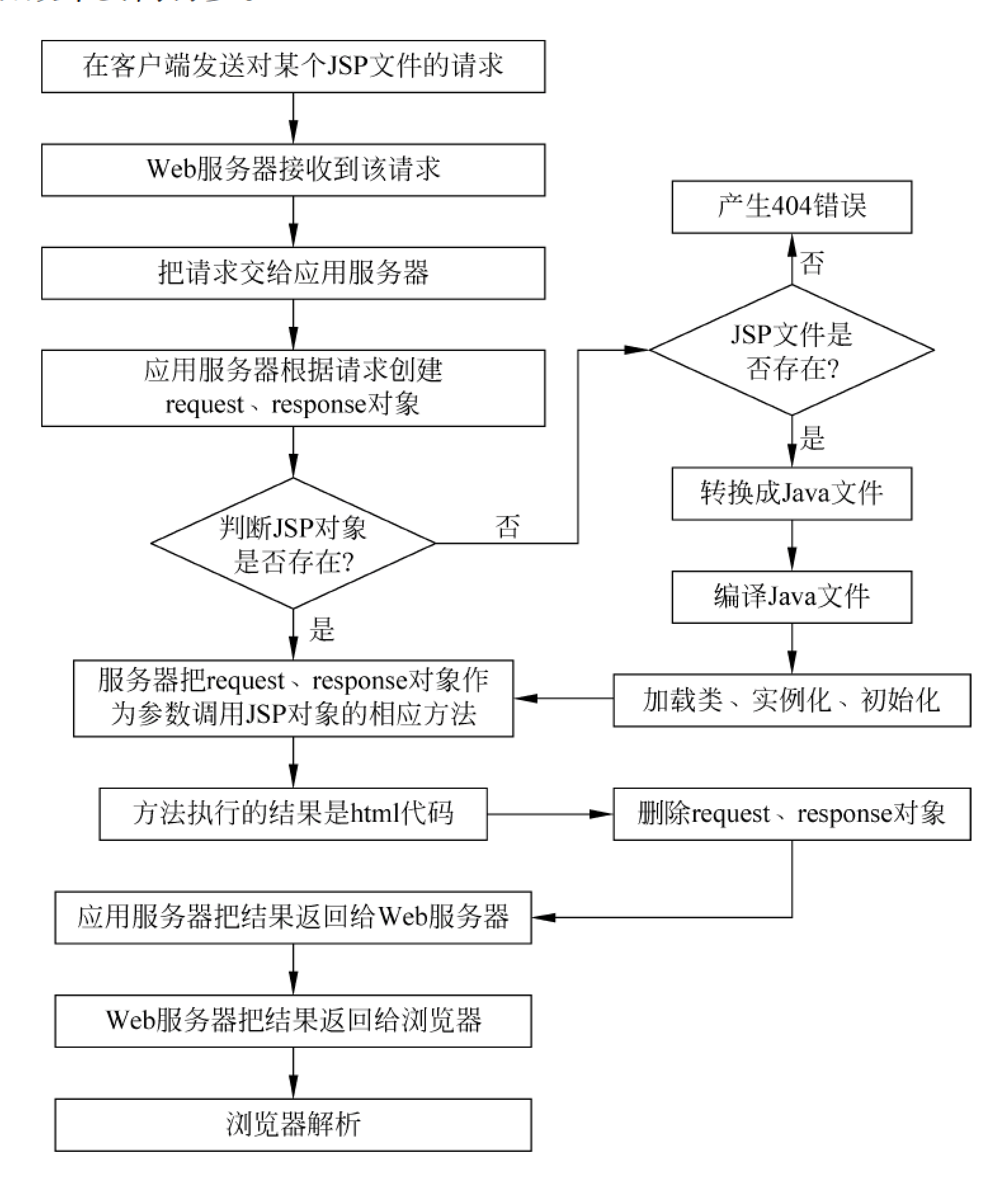


图 1.1 JSP 工作原理

当客户端向一个 JSP 页面发出请求时,Web Container(容器)将 JSP 转化成 Servlet 的源代码(只在第一次请求时),然后编译转化后的 Servlet 并加载到内存中执行,执行的结果 Response(响应)到客户端。JSP 只在第一次执行的时候会转化为 Servlet,以后每次执行 Web 容器都是直接执行编译后的 Servlet,所以 JSP 和 Servlet 只是在第一次执行的时候不一样,JSP 慢一点,以后的执行都是相同的。

下面是被请求的 JSP 文件(sum.jsp)及其在 JSP 引擎里生成的对应的 Java 文件,并

把 JSP 引擎交给客户端显示的内容做了“ *** ”注释。

sum.jsp 源文件如下：

```
<%@ page contentType="text/html;charset=GB2312" %>
<HTML>
<BODY BGCOLOR=cyan>
<FONT Size=1>
<P>这是一个简单的 JSP 页面
     <% int i, sum=0;
         for(i=1;i<=100;i++)
          { sum=sum+i;
          }
      %>
<P>  1 到 100 的连续和是:
<BR>
   <%=sum %>
</FONT>
</BODY>
</HTML>
```

sum.jsp 在 JSP 引擎里生成的对应的 Java 文件如下：

```
package org.apache.jsp;
import javax.servlet. * ;
import javax.servlet.http. * ;
import javax.servlet.jsp. * ;
import org.apache.jasper.runtime. * ;
public class first1 $ jsp extends HttpJspBase {
    static {
    }
    public first1 $ jsp() {
    }
    private static boolean _jspx_inited = false;
    public final void _jspx_init() throws org.apache.jasper.runtime.JspException {
    }
    public void _jspService(HttpServletRequest request, HttpServletResponse  response)
    throws java.io.IOException, ServletException {
        JspFactory _jspxFactory = null;
        PageContext pageContext = null;
        HttpSession session = null;
        ServletContext application = null;
        ServletConfig config = null;
        JspWriter out = null;
        Object page = this;
        String  _value = null;
        try {
            if (_jspx_inited == false) {
                synchronized (this) {
                    if (_jspx_inited == false) {
                        _jspx_init();
                        _jspx_inited = true;
                    }
                }
```

```
                    }
                    _jspxFactory = JspFactory.getDefaultFactory();
                    response.setContentType("text/html;charset=GB2312");
                    pageContext = _jspxFactory.getPageContext(this, request, response, "", true, 8192, true);
                    application = pageContext.getServletContext();
                    config = pageContext.getServletConfig();
                    session = pageContext.getSession();
                    out = pageContext.getOut();
(***)               out.write("\r\n<HTML>\r\n<BODY>\r\n<P>这是一个简单的 JSP 页面\r\n");
                    int i, sum=0;
                        for(i=1;i<=100;i++)
                           { sum=sum+i;
                           }
(***)               out.write("\r\n<P>  1 到 100 的连续和是: \r\n<BR>\r\n   ");
(***)               out.print(sum );
(***)               out.write("\r\n</BODY>\r\n<HTML>\r\n");
                } catch (Throwable t) {
                    if (out != null && out.getBufferSize() != 0)
                        out.clearBuffer();
                    if (pageContext != null) pageContext.handlePageException(t);
                } finally {
                    if (_jspxFactory != null) _jspxFactory.releasePageContext(pageContext);
                }
            }
        }
```

1.5 实践任务：用记事本开发第一个 JSP 应用

1. 任务说明

编写一个简单的 HTML 文件——count.html，这个 HTML 中包含一个 Form 表单，这个表单可以接收用户输入的数字，然后把这个数字发送到 Web 服务器 Tomcat 端的 helloword.jsp。helloword.jsp 根据接收的参数决定显示内容重复的次数，如图 1.2 所示。

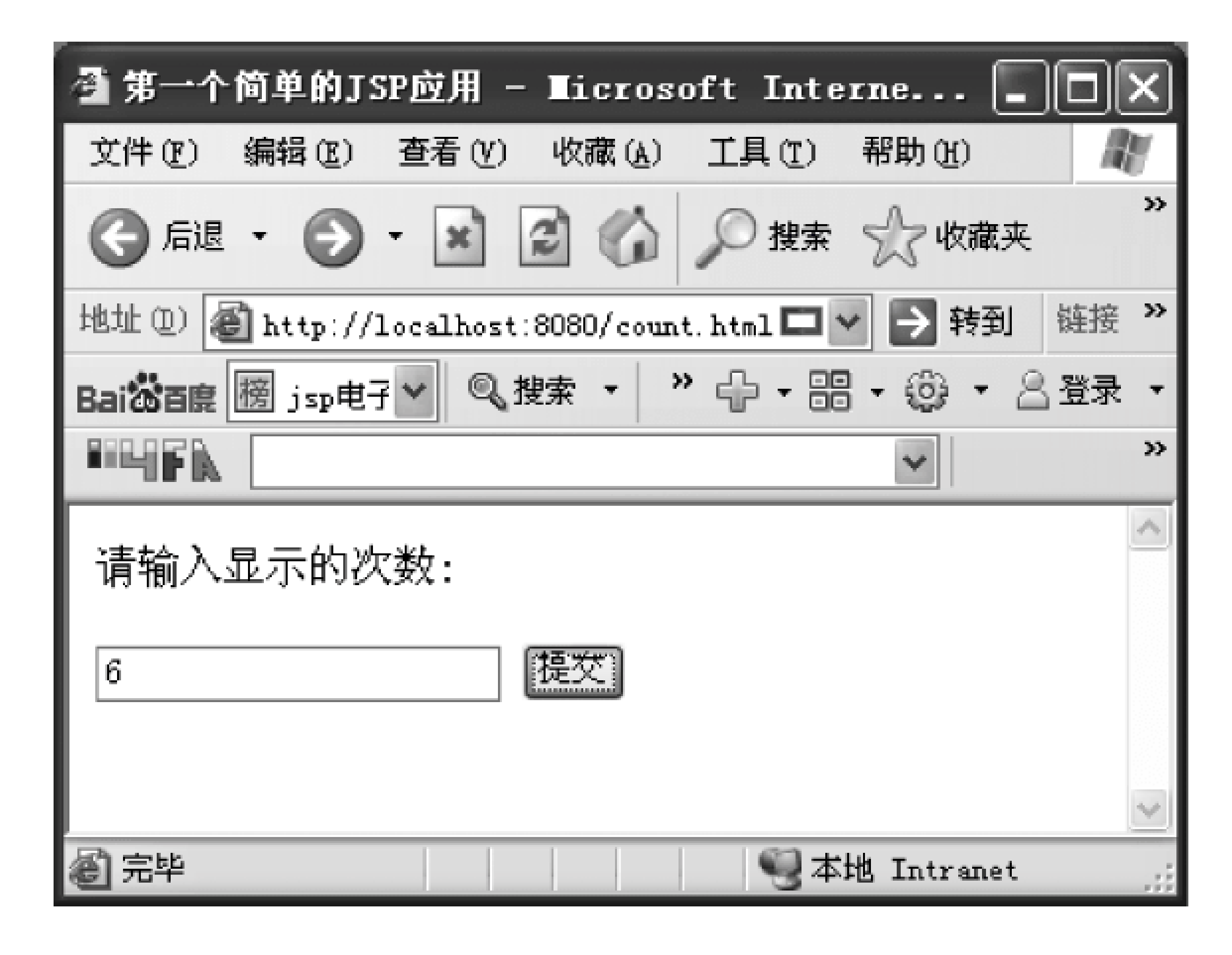


图 1.2 count.html 显示的结果

提交后显示的页面如图 1.3 所示。

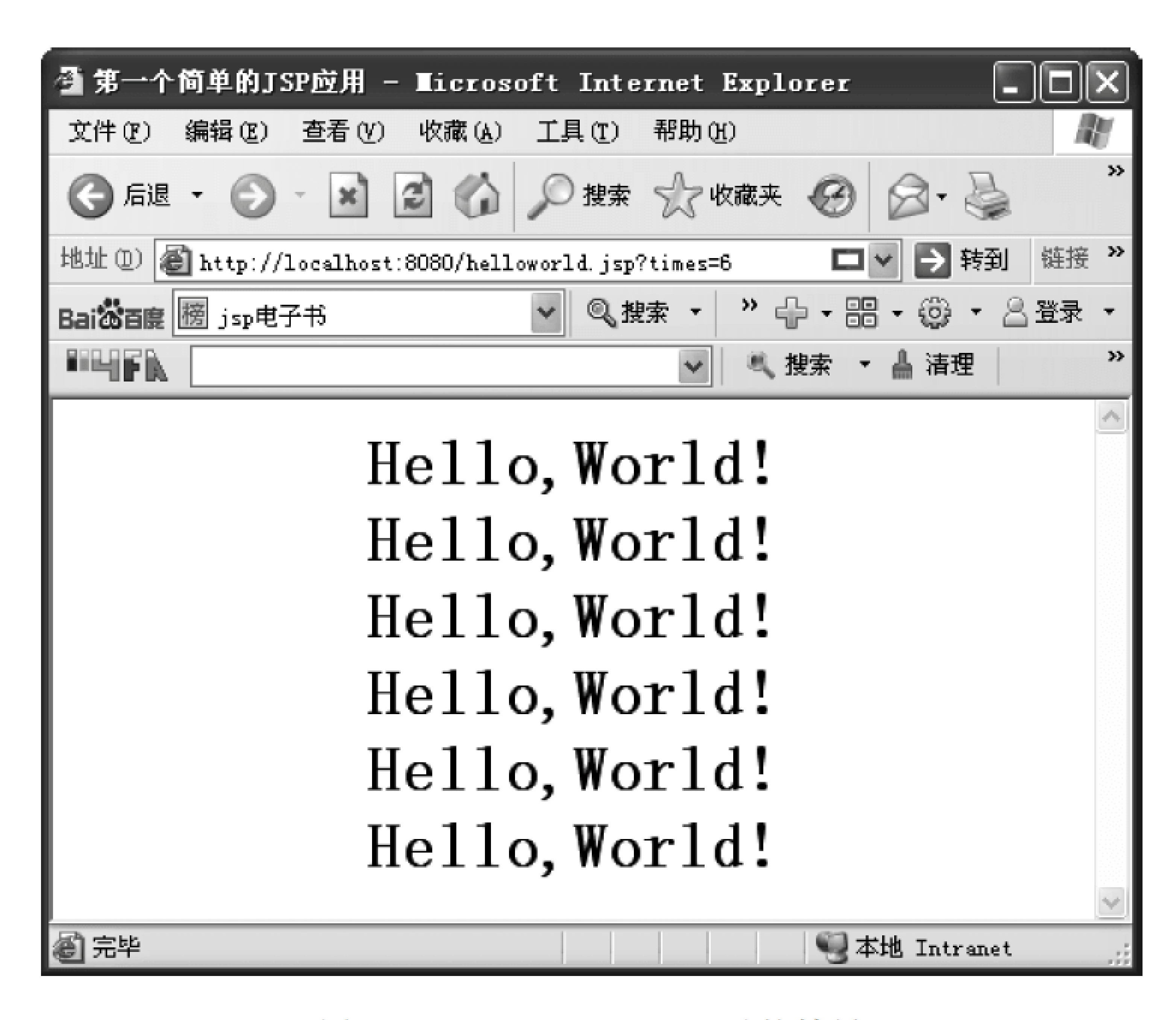


图 1.3　helloworld.jsp 显示的结果

2. 任务实施

任何一个项目都是从开发环境的搭建开始的。要使用 Java 开发 Web 应用,需要 Java 开发环境 JDK 和运行环境 JVM,运行 Java Web 应用程序需要 Web 服务器。因此本任务计划步骤如下。

(1) 安装与配置 JDK 1.6。安装 Java 开发包 J2SDK(Java 2 Software Development Kit)是基于 Java 软件开发的前提。J2SDK 简称 JDK,本书所有程序都基于 JDK 1.6 开发并测试通过。JDK 1.6 在 Sun 的官方网站 http://java.sun.com 上可以下载。

(2) 安装 Web 服务器 Tomcat 6.0。Tomcat 是一个免费的开源的 Web 容器,它是 Apache 基金会的 Jakarta 项目的一个核心项目,由 Apache、Sun 和其他一些公司及个人共同开发而成。由于有了 Sun 的参与和支持,最新的 Servlet 和 JSP 规范总能在 Tomcat 中得到体现。本书所有程序都采用 Tomcat 6.0 作为 Web 服务器。Tomcat 6.0 可以在官方网站 http://tomcat.apache.org 下载。

(3) 开发一个简单的静态页面 count.html。由于本任务是第一个入门的 JSP 应用,因此没有使用 IDE 集成开发环境(比如 Myeclipse、Jbuilder 等)。后面的章节会介绍 IDE 集成开发环境的使用,以便提高开发效率。创建一个记事本,录入以下代码,并另存为 count.html。

```
<html>
    <head>
    <title>第一个简单的 JSP 应用</title>
    </head>
```

```
    <body>
    <p>请输入显示的次数:</p>
    <form method=get action="helloworld.jsp">
    <input type="text" name="times">
    <input type=submit value="提交">
    </form>
    </body>
</html>
```

(4) 开发一个简单的JSP动态页面helloworld.jsp。创建一个记事本,录入以下代码,并另存为helloworld.jsp。

```
<%@ page language="java" contentType="text/html;charset=gb2312"%>
<html>
    <head>
    <title>第一个简单的JSP应用</title>
    </head>
    <body>
    <center>
    <h1>
    <%
        //通过jsp内置对象request的getParameter()方法,获得参数名为times的表单数据并
        //转换为整型数据存储到整型变量times中
        int times=Integer.parseInt(request.getParameter("times"));
        for(int i=0;i<times;i++){
            //通过jsp的内置对象out的println()方法,输出Hello,World!到页面
            out.println("Hello,World!");
            out.println("<br>");
        }
    %>
    </h1>
    </center>
    </body>
</html>
```

(5) 部署发布JSP动态页面。

① 将count.html和helloworld.jsp两个文件分别复制到tomcat安装目录Tomcat6.0\webapps\ROOT下。

② 启动tomcat,注意端口号,默认为8080。

③ 启动浏览器,并输入http://localhost:8080/count.html。

④ 正确出现count.html页面后,在文本框中输入数字后提交,即进入helloworld.jsp页面,测试完成。

3. 任务总结

该实践任务是开发第一个JSP应用。所以,首先需要安装、配置JSP的开发、运行环境。开发、运行环境测试好以后,就可以着手开发页面了。最后,部署、发布页面并进行测试。通过本任务,主要了解一下JSP应用的开发、部署及测试。

4. 拓展任务

编写一个简单的HTML文件——count.html,这个HTML中包含一个Form表单,

这个表单可以接收用户输入的数字，然后把这个数字发送到 Web 服务器 Tomcat 端的 date.jsp，date.jsp 根据接收的参数决定显示系统当前时间的次数，并要求奇数次为一种颜色、偶数次为另一种颜色。

1.6　实训

实训题目：开发第一个简单的 JSP 应用。

实训目标：要求完成 Java Web 开发环境的建立，包括软件下载、安装和配置系统环境变量，并要编写简单的 JSP 进行测试。

1.7　本章小结

本章主要介绍了什么是 JSP、什么是动态网页技术，并从技术角度和商业角度对比较常用的动态网页技术如 JSP、ASP、PHP 等进行了阐述比较，还介绍了 Java、Servlet、JSP 之间的联系及 JSP 的运行原理。最后，通过“实践任务：同记事本开发第一个 JSP 应用”，详细讲解了 JSP 应用的开发、测试环境的搭建与运行。

第2章

搭建开发环境

对于初学者,如何搭建良好的开发运行环境是第一个面临的实际问题。本章将对 Java Web 应用的开发运行环境的安装与配置进行详细的讲解,图文并茂,以指导读者一步一步操作,直到完成。

本章要点:

- JDK 1.6 的安装与配置
- Web 服务器 Tomcat 6.0 的安装与配置
- 数据库服务器 MySQL 5.0 的安装
- MyEclipse 8.0 集成开发环境的安装

2.1 安装和配置 JDK

安装 Java 开发包 J2SDK(Java 2 Software Development Kit)是基于 Java 软件开发的前提。J2SDK 简称 JDK,本书所有程序都基于 JDK 1.6 开发并测试通过。JDK 1.6 在 Sun 的官方网站 http://java.sun.com 上可以下载。下面具体讲解 JDK 1.6 在 Windows 操作系统上的安装与配置步骤。

2.1.1 JDK 的安装

(1) 双击运行 j2sdk-1.6.exe 文件,开始安装,如图 2.1 所示。



图 2.1　运行 j2sdk-1.6.exe

(2) 选择安装路径及安装内容,如图 2.2 所示。

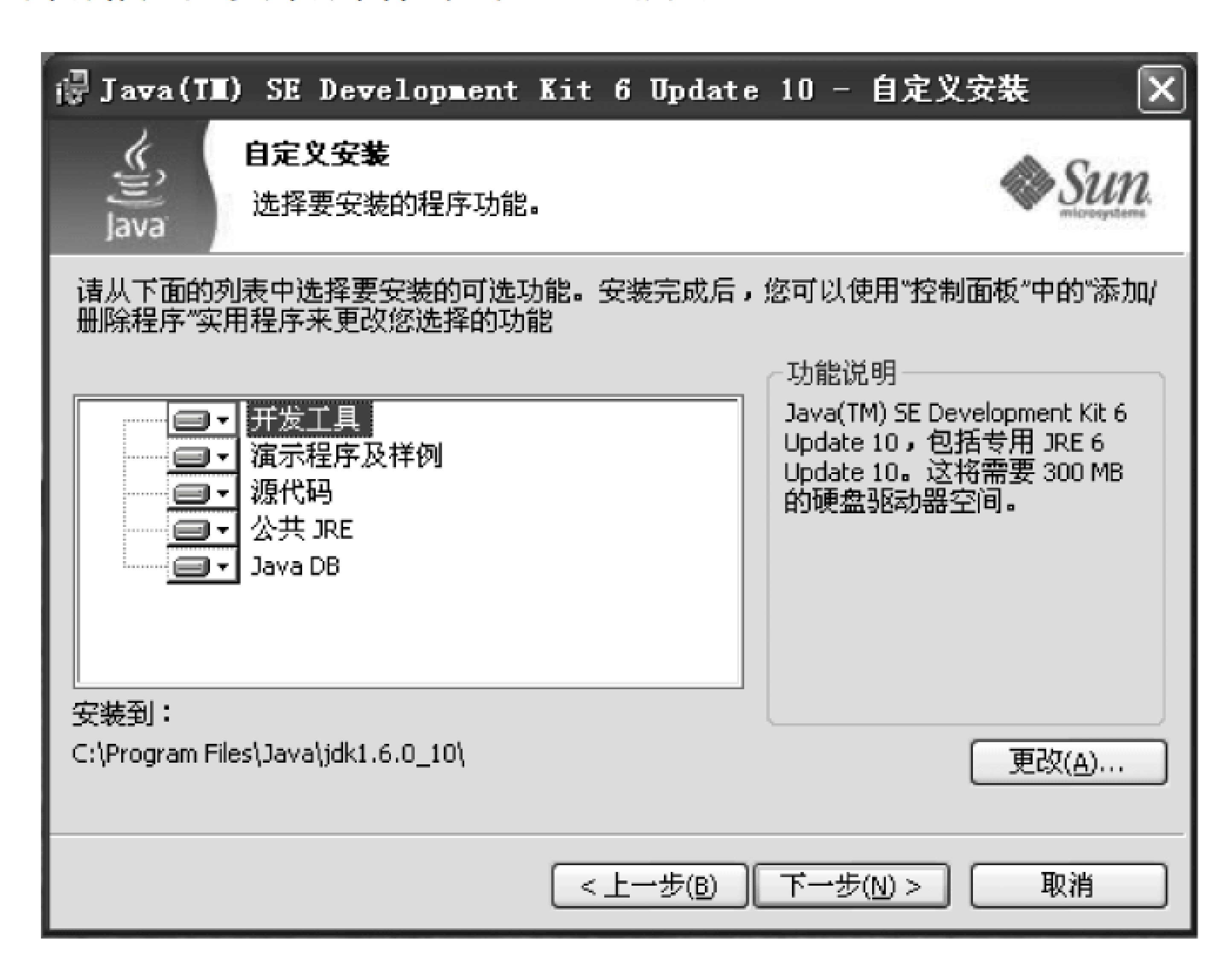


图 2.2　选择安装路径及安装内容

(3) 安装完成,如图 2.3 所示。



图 2.3　安装完成

2.1.2　JDK 的配置

安装好 JDK 后,还需要在环境变量中进行对应的配置。下面以 Windows 操作系统为例进行配置。主要进行以下的配置:设置 JAVA_HOME 环境变量;设置 CLASSPATH 环境变量;设置 PATH 环境变量的值。

(1) JAVA_HOME 表示 JDK 的安装目录,其他的应用程序如果需要使用 Java 运行

环境，首先获得 JAVA_HOME 变量的信息。在“我的电脑”上右击，在弹出的快捷菜单中选择“系统属性”命令，选择“高级”|“环境变量”，对其进行配置，如图 2.4 所示。

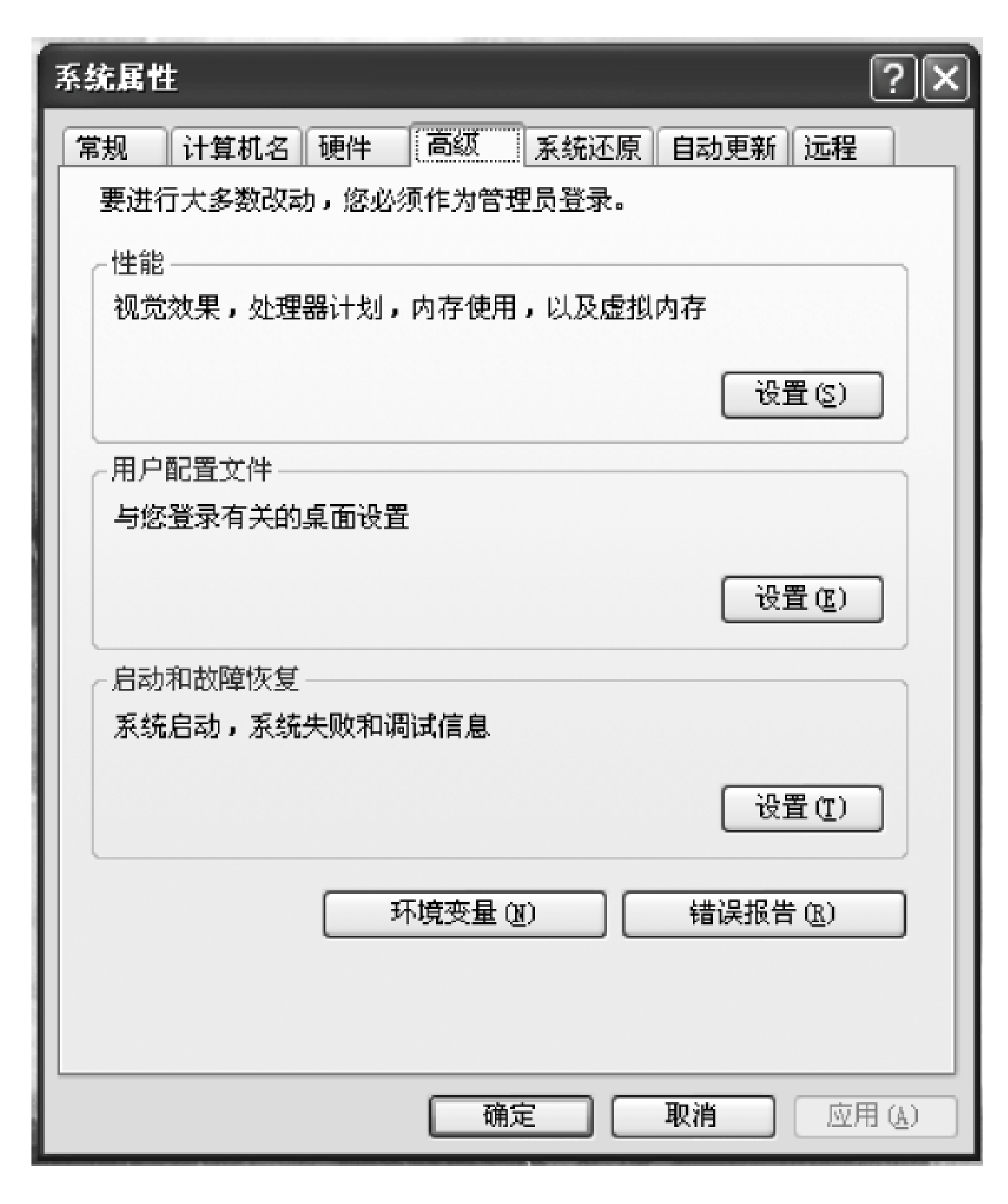


图 2.4 设置系统的环境变量

(2) 在“系统变量”中，新建变量 JAVA_HOME，变量值为 C:\Program Files\Java\jdk1.6.0_10，如图 2.5 所示。



图 2.5 新建变量 JAVA_HOME

(3) 新建系统变量 path，变量值为%JAVA_HOME%\bin，如图 2.6 所示。



图 2.6 新建系统变量 path

(4) 新建系统变量 classpath，变量值为 .;%JAVA_HOME%\lib\dt.jar;%JAVA_HOME%\lib\tools.jar，如图 2.7 所示。

至此，我们完成了 JDK 的安装及配置。

新建系统变量
变量名(N): classpath
变量值(V): .;%JAVA_HOME%\lib\dt.jar;%JAVA_HOME%
确定　取消

图 2.7　新建系统变量 classpath

(5) 测试 J2SDK。单击"开始"|"运行"|cmd 命令，在弹出的 DOS 窗口中输入 javac 并按 Enter 键，如图 2.8 所示。



图 2.8　测试成功

2.2　安装和配置 Tomcat

Tomcat 是一个免费的，开源的 Web 容器。它是 Apache 基金会的 Jakarta 项目的一个核心项目，由 Apache、Sun 和其他一些公司及个人共同开发而成。由于有了 Sun 的参与和支持，最新的 Servlet 和 JSP 规范总能在 Tomcat 中得到体现。本书所有程序都采用 Tomcat 6.0 作为 Web 服务器。Tomcat 6.0 可以在官方网站 http://tomcat.apache.org 下载。

在安装 Tomcat 6.0 之前，要确保 JDK 已经成功安装，并正确配置 JAVA_HOME 环

境变量,因为 Tomcat 是基于 JRE(Java Runtime Environment)工作的。下面具体讲解 Tomcat 6.0 在 Windows 操作系统中的安装步骤。

2.2.1 安装 Tomcat

(1) 运行 Tomcat-6.0.exe 文件,开始安装,如图 2.9 所示。

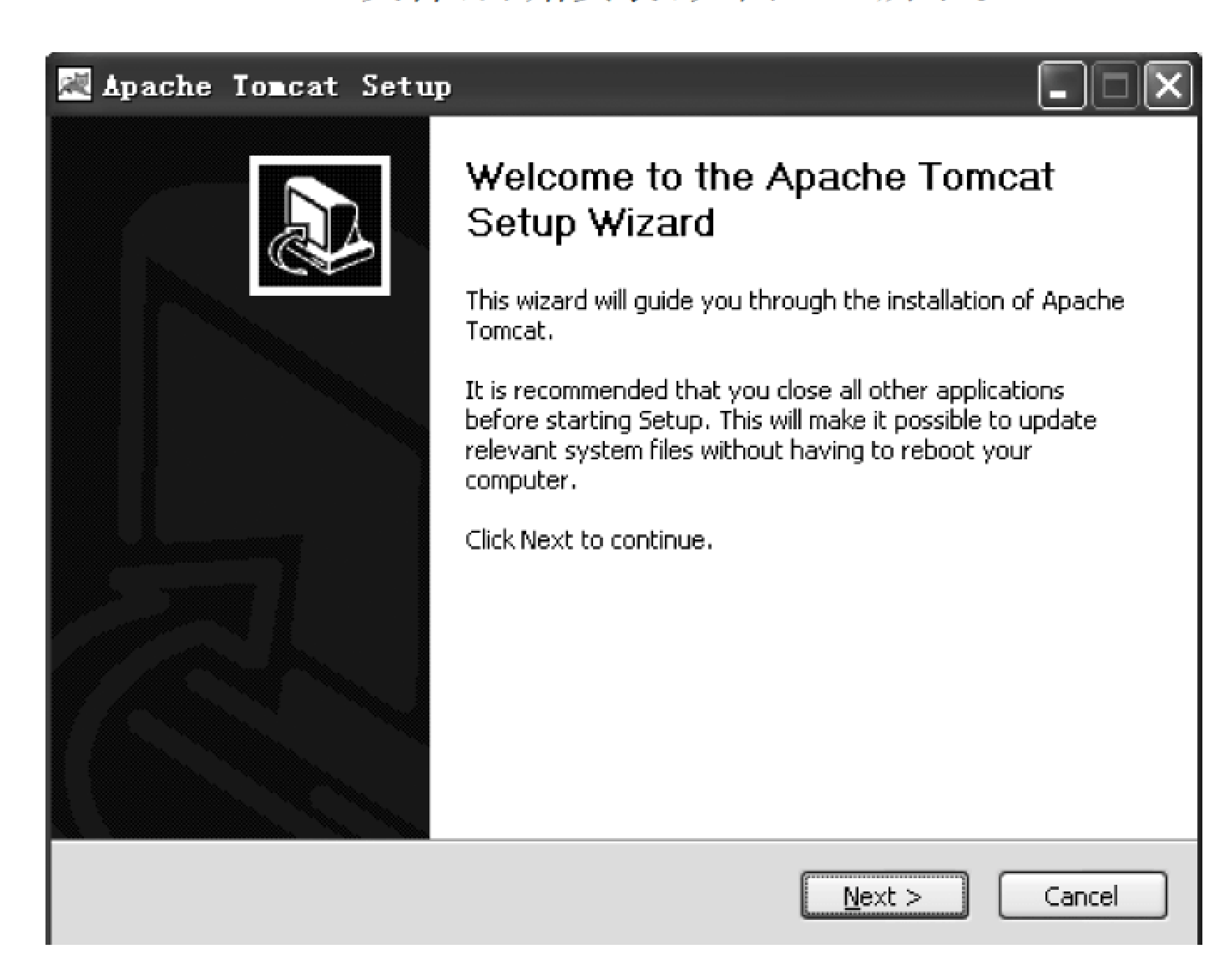


图 2.9 开始安装

(2) 选择安装内容及安装路径,如图 2.10 和图 2.11 所示。

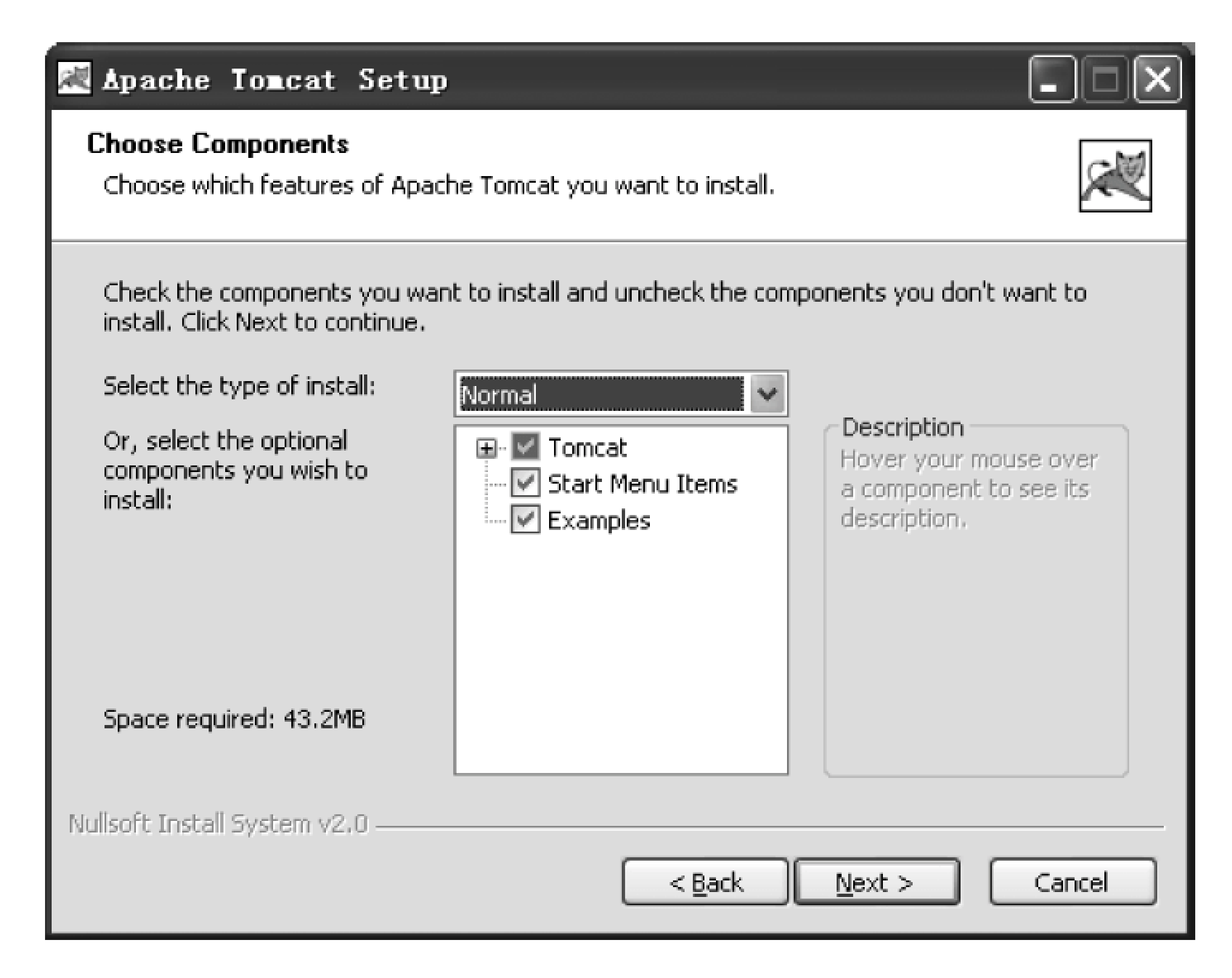


图 2.10 选择安装内容

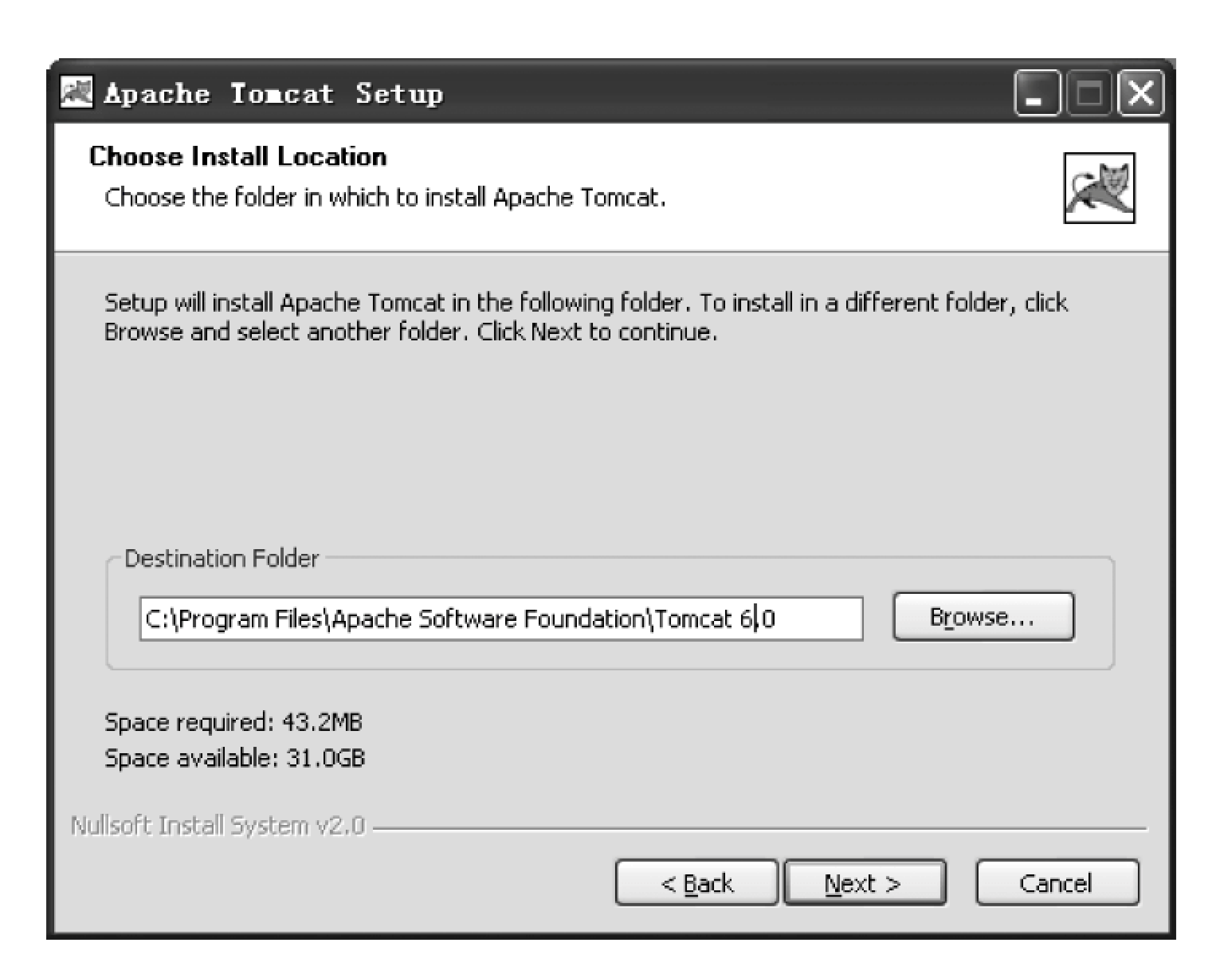


图 2.11 选择安装路径

(3) 设置 Connector Port 和 Administrator Login,如图 2.12 所示。

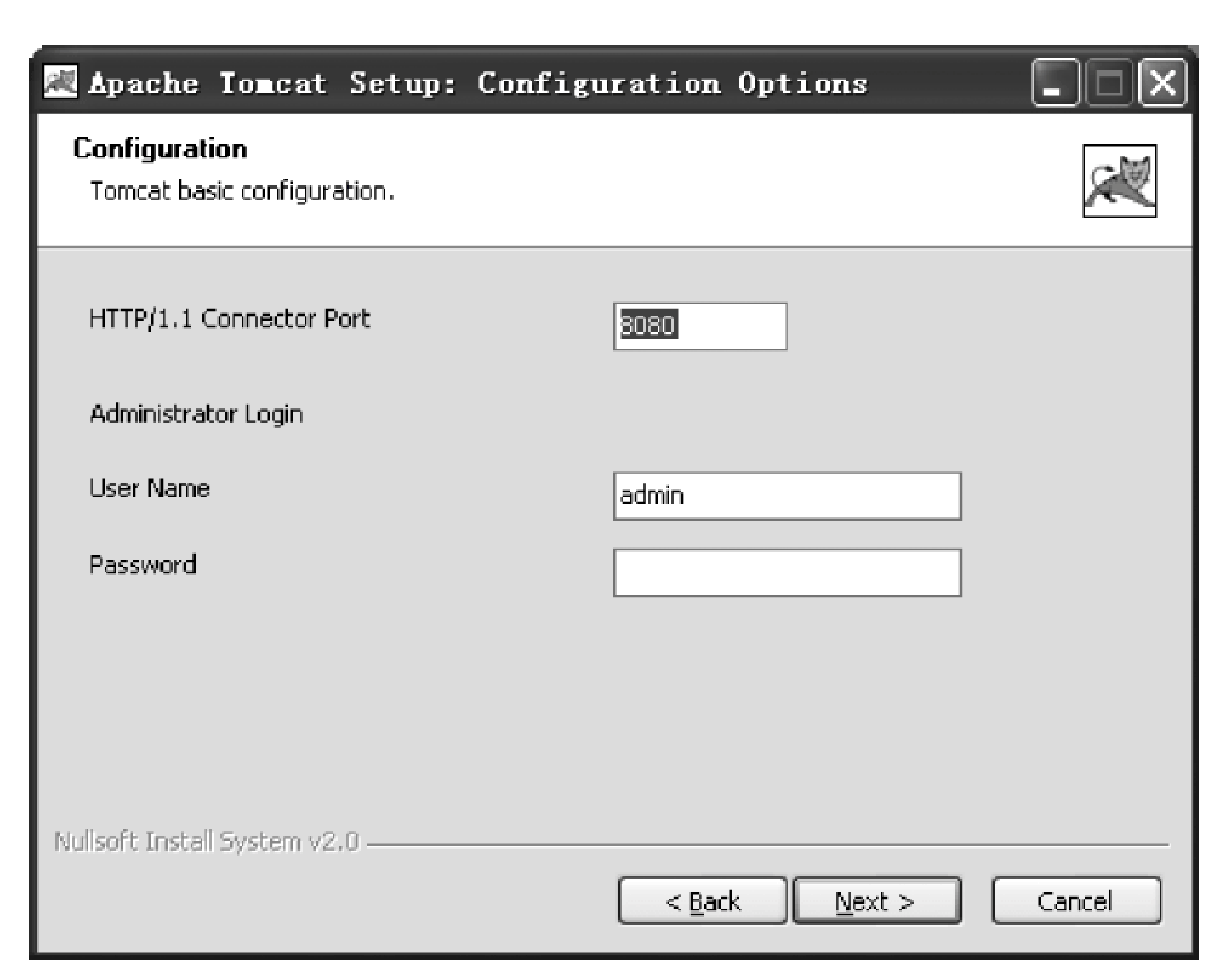


图 2.12 设置 Connector Port 和 Administrator Login

(4) 设置 Tomcat 使用的 JVM,如图 2.13 所示。

(5) 完成安装,如图 2.14 所示。

2.2.2 测试安装是否成功

下面测试 Tomcat。首先启动 Tomcat,然后打开浏览器,输入 http://localhost:

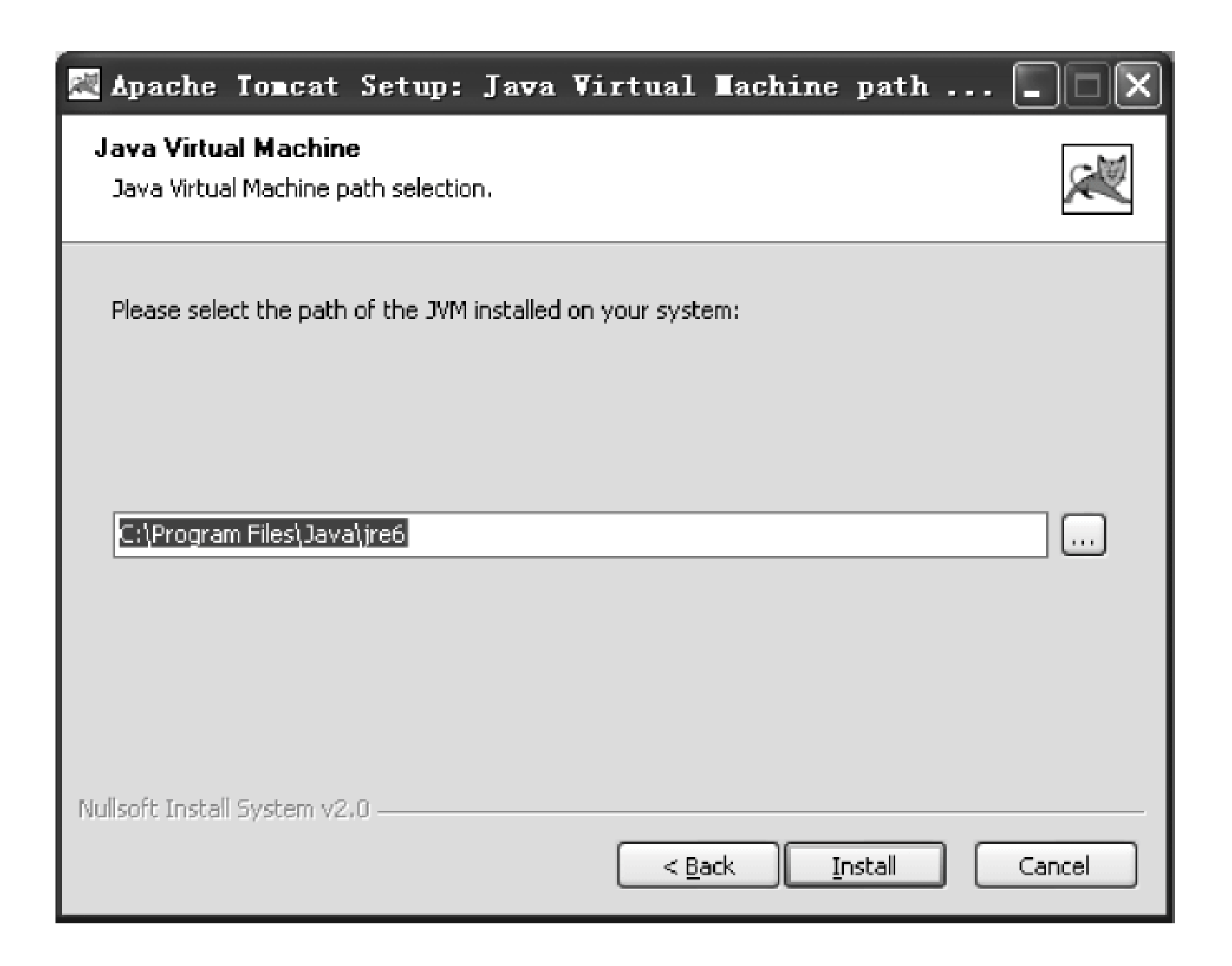


图 2.13　设置 Tomcat 使用的 JVM

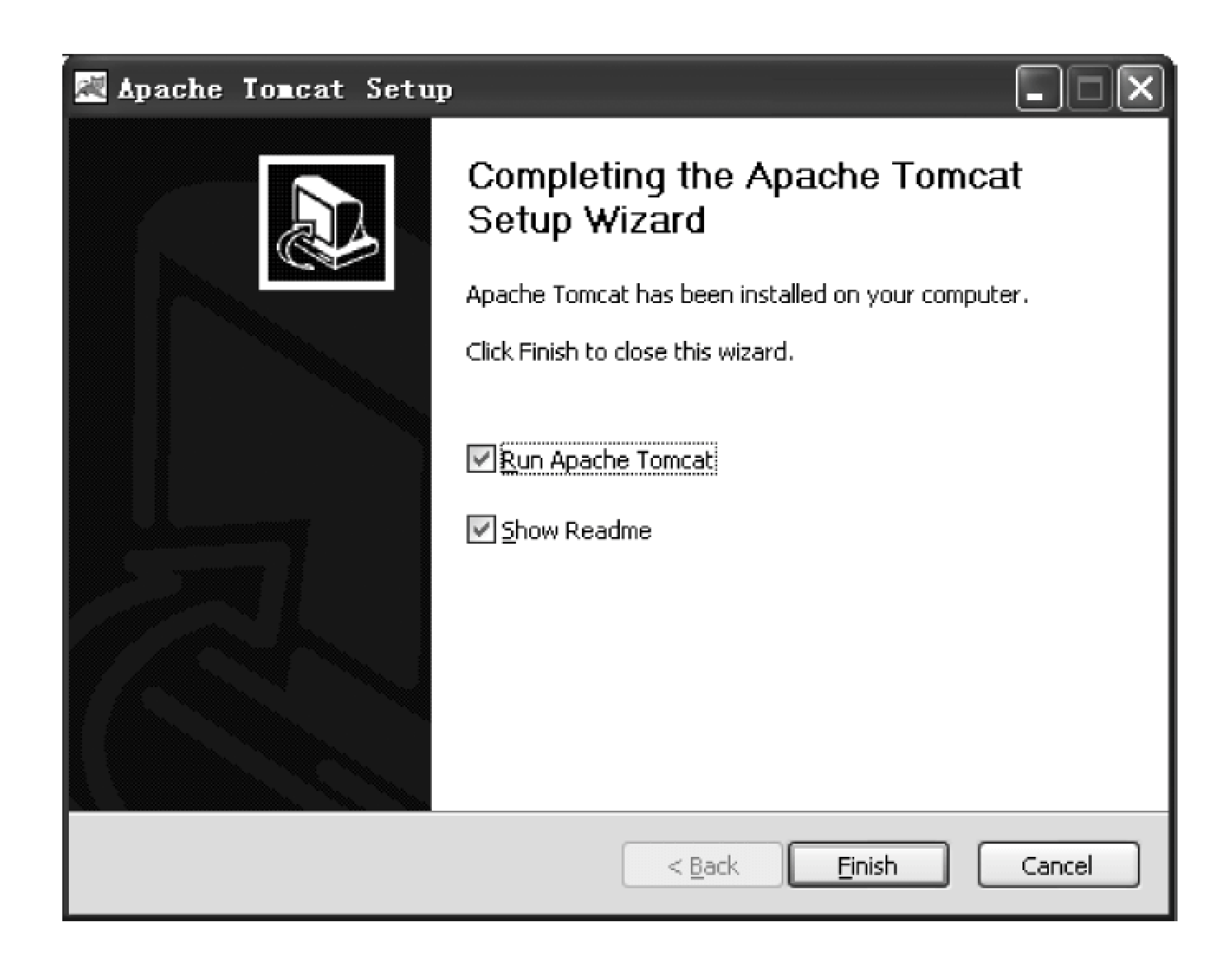


图 2.14　完成安装

8080,之后按 Enter 键,如图 2.15 所示。

2.2.3　测试第一个 JSP 页面

编写一个简单的 JSP,如例 2.1 所示。

【例 2.1】　用于测试 Web 服务器的 JSP 页面(ch2/test.jsp)。

```
<%@ page contentType="text/html;charset=GB2312"%>
<%@ page import="java.util.*"%>
```

```
<HTML>
<BODY>
<P> 现在的时间是:
    <% Date date=new Date();%>
<BR>
<%=date%>
</BODY>
</HTML>
```

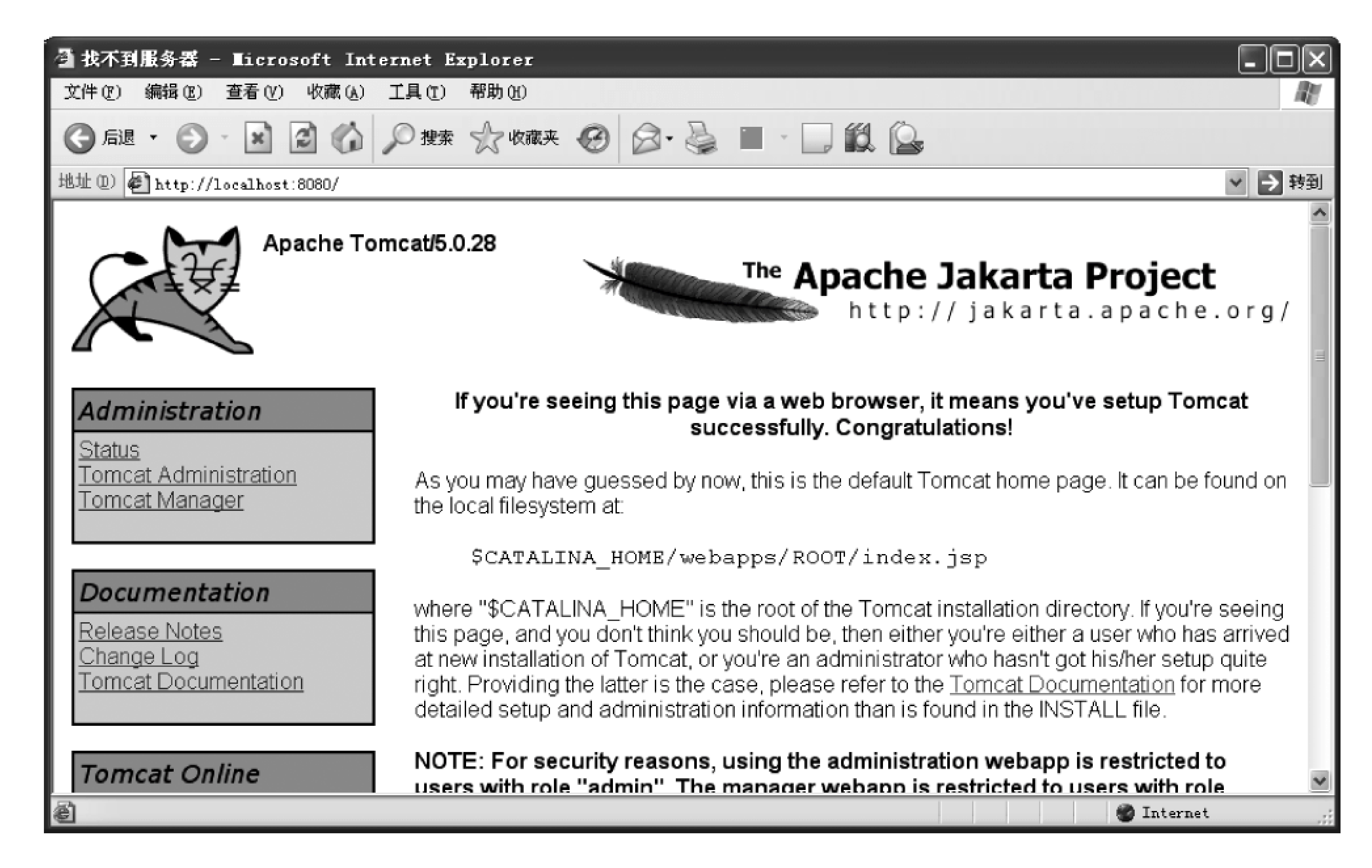


图 2.15　测试成功

把 ch2 目录复制到%TOMCAT_HOME%\webapps\目录下，启动 Tomcat。在浏览器输入 http://127.0.0.1:8080/ch2/test.jsp，如果出现的结果如图 2.16 所示，那么恭喜你，第一个 JSP 页面的编写、部署就成功了！

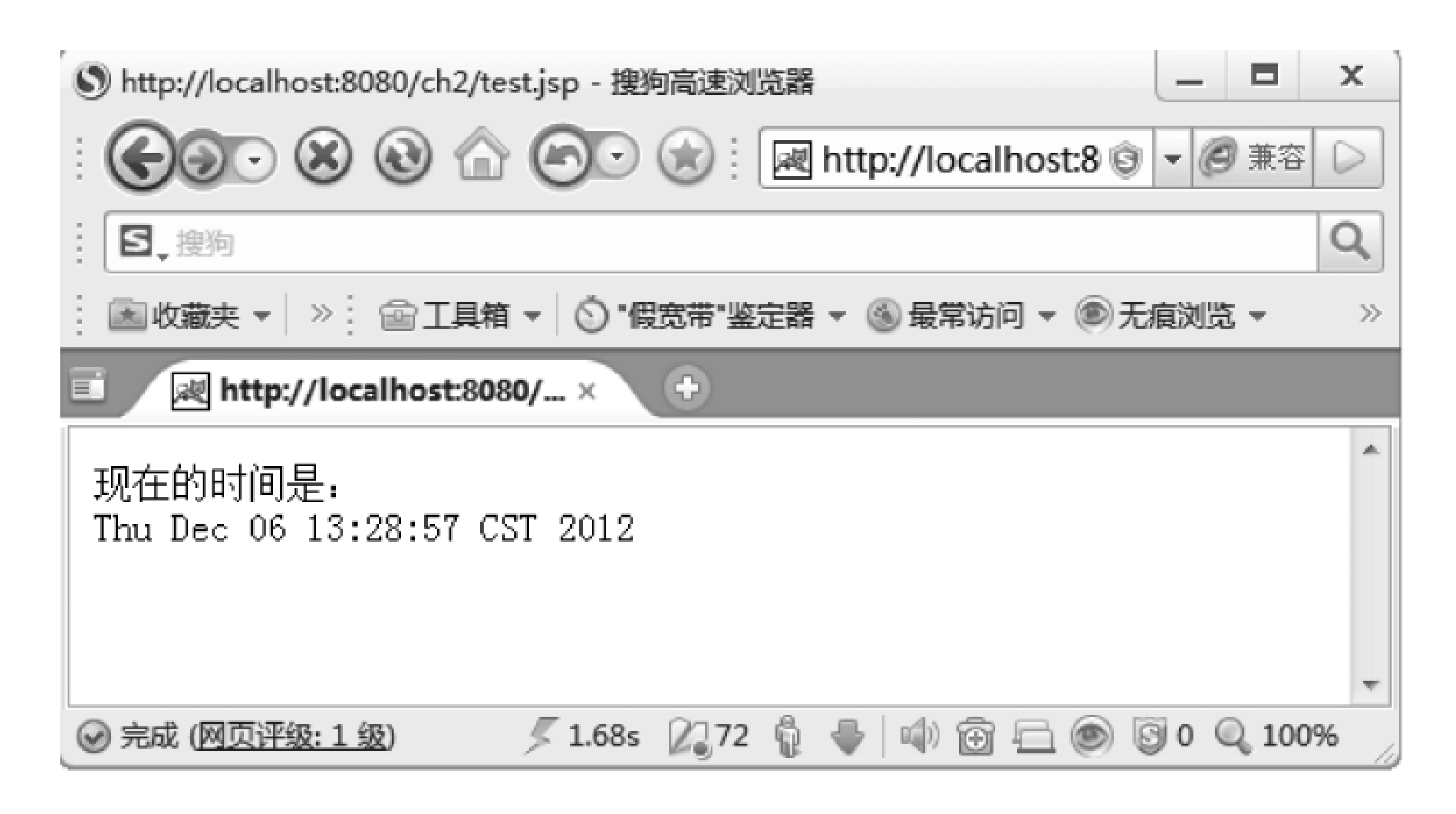


图 2.16　测试 JSP 页面

2.2.4 配置 Tomcat 服务器

1. 通过 server.xml 配置 Tomcat

Tomcat 最重要的配置文件为 server.xml。server.xml 中的元素大致可以分为以下 4 类。

(1) 顶层类元素：<Server>元素与</Server>元素，位于整个配置文件的顶层。

(2) 连接器元素：是客户和服务(容器类元素)间的通信接口，负责接收客户请求以及为客户返回响应结果，主要有<Connector>元素。

(3) 容器类元素：负责处理客户请求并且生成响应结果，主要有<Engine>元素、<Host>元素和<Context>元素。

(4) 嵌套类元素：可以加入容器中的元素，主要有<logger>元素、<Value>元素、<Realm>元素等。

Server 元素：根元素，它可以包含一个或多个 Service 实例。

Service 元素：由一个或多个 Connector 与 Engine 组成。

Connector 元素：是一个 Tomcat 与客户端的连接。

Engine 元素：可以配置多个虚拟机主机 Host，并将请求配置到具体的 Host 上。

Host 元素：代表一个虚拟主机，其下可以配置多个 Web 应用。

Context 元素：代表一个 web 应用。

各元素的属性如下所述。

(1) Server 元素。

port：设置负责监听关闭 Tomcat 请求的端口。

shutdown：设置一个向端口发送的命令字符串。

(2) Service 元素。

Name：设置 Service 实例的名字。

(3) Connector 元素。

Port：设置一个监听来自客户端请求的端口。

maxThreads：设置可创建用于处理请求的最大线程数。

enableLookups：该属性直接影响 request.getRemoteHost()方法的返回结果，当设置为 true 时，将通过查询 DNS 取得远程客户端的实际主机名；当设置为 false 时，则直接返回其 IP。

redirectPort：设置服务器在处理 Http 请求时受到 SSL 传输请求后的重定向端口。

acceptCount：设置请求队列的大小。

connectionTimeout：设置连接超时的毫秒数，如果为－1，表示不限制建立客户的连接时间。

(4) Engine 元素。

Name：设置 Engine 实例的名字。

defaultHost：设置一个处理请求的默认虚拟主机名。

(5) Host 元素。

name：设置虚拟主机名。

appBase：设置存放应用程序的根目录。

unpackWARS：当设置为 true 时，tomcat 会自动解压 WAR 文件。

autoDeploy：当设置为 true 时，tomcat 服务处于运行状态，能够监听 appBase 下的文件，　如果有新 Web 应用加入，则会自动发布这个 Web 应用。

(6) Context 元素。

docBase：设置 Web 应用或 WAR 文件的存放路径。

path：设置访问 Web 应用入口 URL。

reloadable：当设置为 true 时，tomcat 服务器在运行状态下会监听 WEB-INF/classes 与 Web-INF/lib 目录下 class 文件的变化情况，如果监听到 class 文件被更新，服务器就重新加载 Web 应用。

2. Web 应用的目录结构

根据 Java EE 规范要求，Java Web 应用必须具有固定的目录结构，通常我们需要建立一个 Web 应用的根目录。假设在 $CATALINA_HOME/webapps 下有 helloapp 的 Web 应用，其目录结构会如下所示。

/helloapp：Web 应用的根目录，所有的 JSP 文件和 HTML 文件都在此目录下。

/helloapp/WEB_INF：存放该 Web 应用发布时的描述文件 web. xml。

/helloapp/WEB_INF/class：存放各种 class 文件，Servlet 文件也存放于此目录下。

/helloapp/WEB_INF/lib：存放各种 Web 应用所需要的 jar 文件，比如可以存放 JDBC 驱动程序的 JAR 文件。

3. web. xml 配置详解

下面是 web. xml 的基本结构。

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee" xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance" xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun
.com/xml/ns/j2ee/web-app_2_4.xsd" version="2.4">
<display-name>WebModule1</display-name>
</web-app>
```

其中，<display-name>标签是 WebModule 的命名。如果需要加入一个 Servlet，就应加入下面这段标签。

```
<servlet>
<servlet-name>servlet1</servlet-name>
<servlet-class>untitled1.Servlet1</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>servlet1</servlet-name>
<url-pattern>/servlet1</url-pattern>
</servlet-mapping>
```

其中，<servlet-name>表示 Servlet 的命名；<servlet-class>表示此 Servlet 所在哪个

包、是哪个类；<url-pattern>是与过滤器配合使用的，过滤器是 Filter，其配置也是在这个文件里设置。

4. 如何修改 Tomcat 的默认端口号

Tomcat 默认的端口号为 8080，如果想修改端口号，只需要在 service. xml 文件中找到以下代码。

```
<Connector port="8080" protocol="HTTP/1.1"
connectionTimeout="20000"
redirectPort="8443" />
```

把它改为：

```
<Connector port="8888" protocol="HTTP/1.1"
connectionTimeout="20000"
redirectPort="8443" />
```

保存后，重新启动 Tomcat 即可。

5. 如何配置虚拟主机

虚拟主机是一种在一个 Web 应用服务器上服务多个域名的机制，对每个域名而言，好像都可以独享整个主机。在 Tomcat 中，可以通过 server. xml 中添加 Host 元素来实现，每个 Host 元素必须包含一个或多个 context 元素，且要包含一个默认的 context，此默认的 context 的访问路径需要设置为空，如下所示。

```
…
<Host name="Site1"    appBase="Host1">
<Context    path=""    docBase="."/>
</Host>
…
```

6. 部署 Web 应用

方法一：可以将应用程序直接放到 Webapps 目录下，Tomcat 的 Webapps 目录是 Tomcat 默认的应用目录，当服务器启动时，会加载这个目录下的所有应用。

方法二：在 Tomcat 的配置文件中，一个 Web 应用就是一个特定的 Context，可以通过在 server. xml 中新建的 Context 里部署一个 JSP 应用程序。打开 server. xml 文件，在 Host 标签内建一个 Context，内容如下：

```
<Context path="/myapp" reloadable="true" docBase="D:\myapp" workDir="D:\myapp\work"/>
```

其中，path 是虚拟路径；docBase 是 JSP 应用程序的物理路径；workDir 是这个应用的工作目录，存放这个应用运行时生成的相关文件。

2.3 安装和配置 MySQL

MySQL 是一个小型关系型数据库管理系统，开发者为瑞典 MySQL AB 公司。目前 MySQL 被广泛地应用在 Internet 上的中小型网站中。由于其体积小、速度快、总体拥有

成本低,尤其是开放源码这一特点,所以许多中小型网站为了降低网站总体拥有成本而选择了 MySQL 作为网站数据库。下面具体讲解 MySQL 5.1 在 Windows 操作系统中的安装步骤。

2.3.1　安装 MySQL

(1) 双击 MySQL 安装程序,如图 2.17 所示。单击 Next 按钮,开始安装。

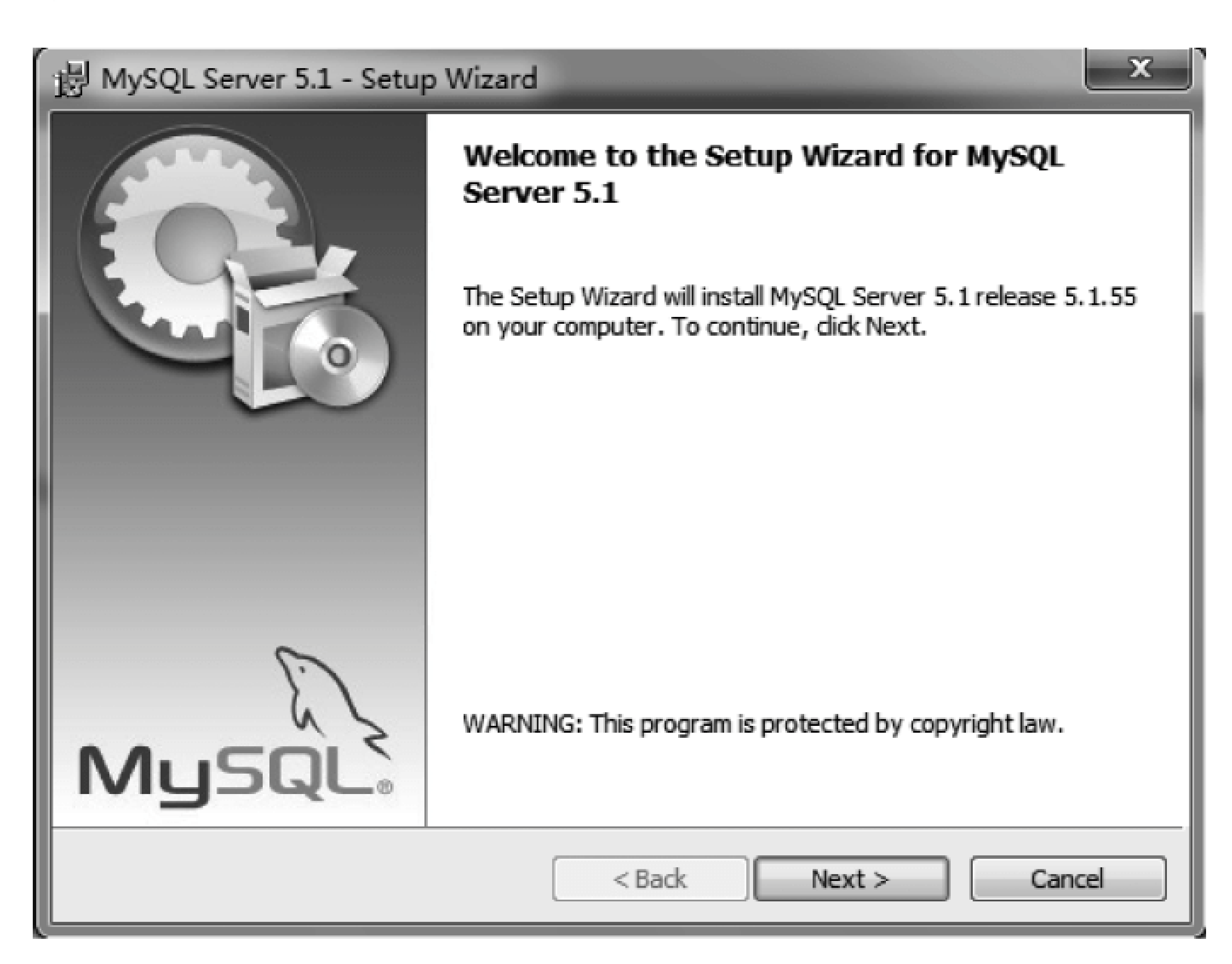


图 2.17　安装 MySQL 5.1 初始窗口

(2) 选择接受许可协议,如图 2.18 所示,单击 Next 按钮。

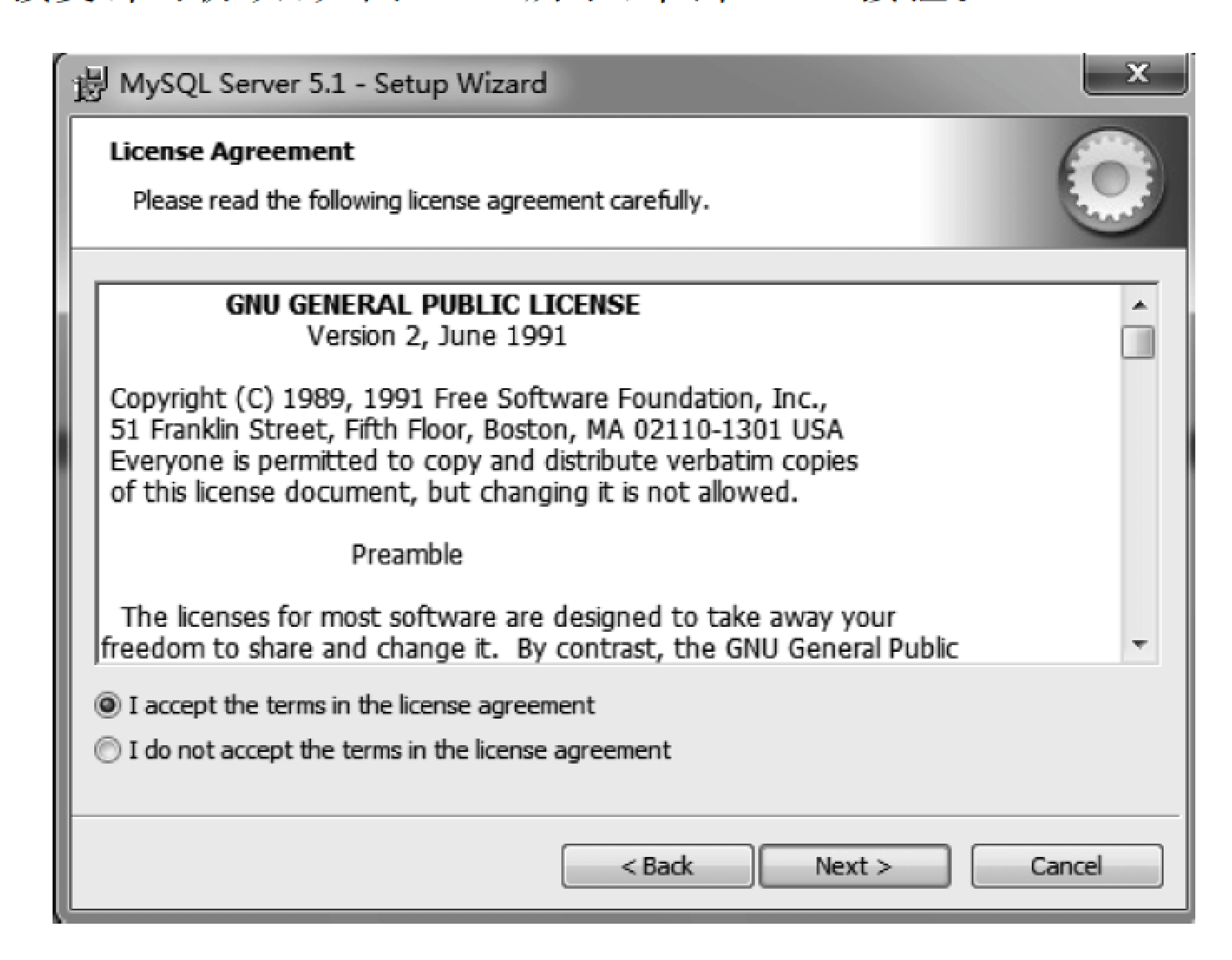


图 2.18　接受许可协议界面

(3) 进入安装窗口选择 Custom,然后单击 Next 按钮,进入 MySQL 的安装向导中,如图 2.19 所示,单击 Install 按钮。

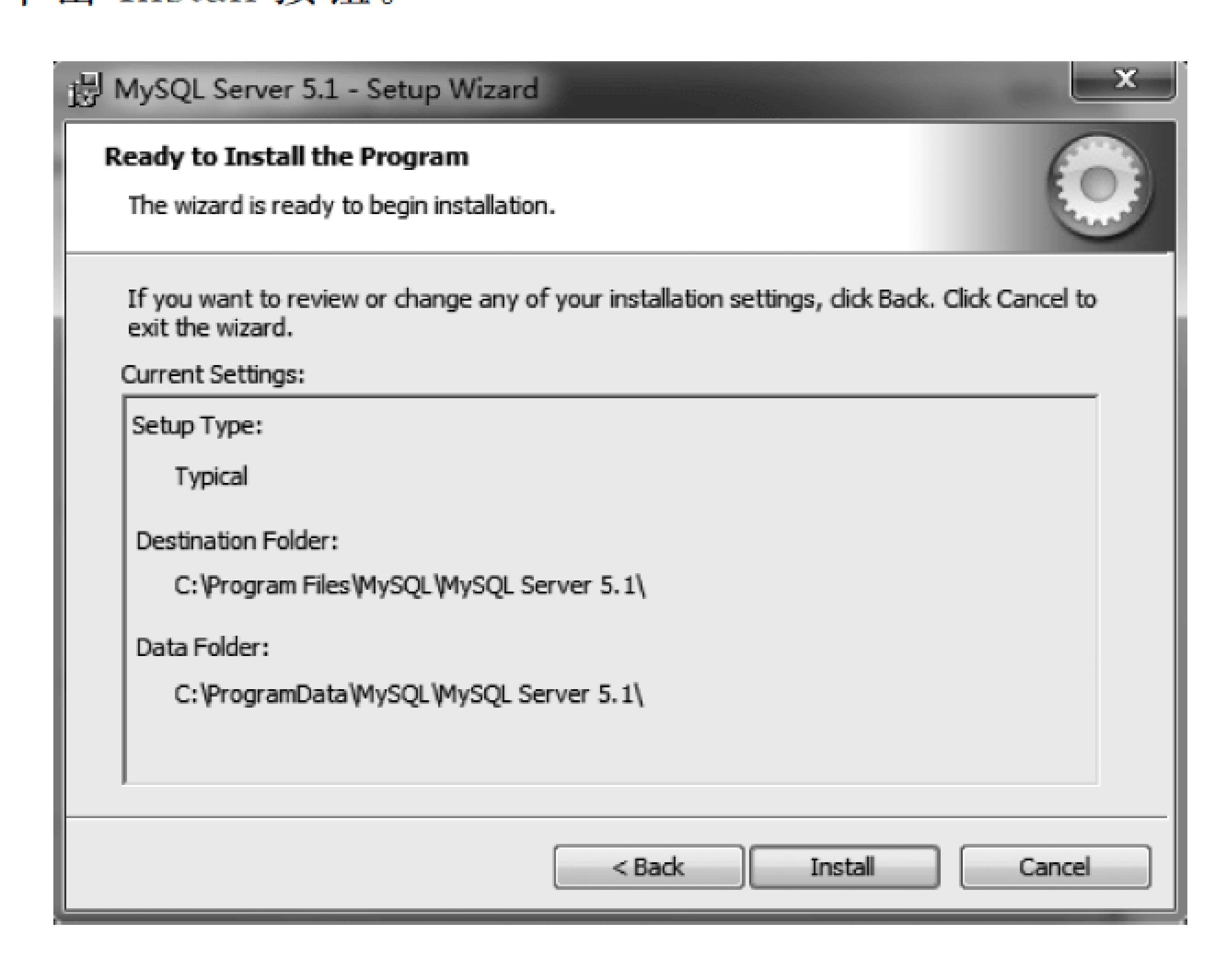


图 2.19 选择安装路径的界面

(4) 进入 MySQL 的安装进程,继续单击 Next 按钮,进入完成界面,如图 2.20 所示。



图 2.20 安装进程结束界面

2.3.2 配置 MySQL

(1) 选择配置方式 Standard Configuration,单击 Next 按钮,如图 2.21 所示。

(2) 将 MySQL 服务器作为 Windows 的服务器,设置 root 管理员账号和密码,如图 2.22 所示。

(3) 单击 Finish 按钮,完成安装,如图 2.23 所示。

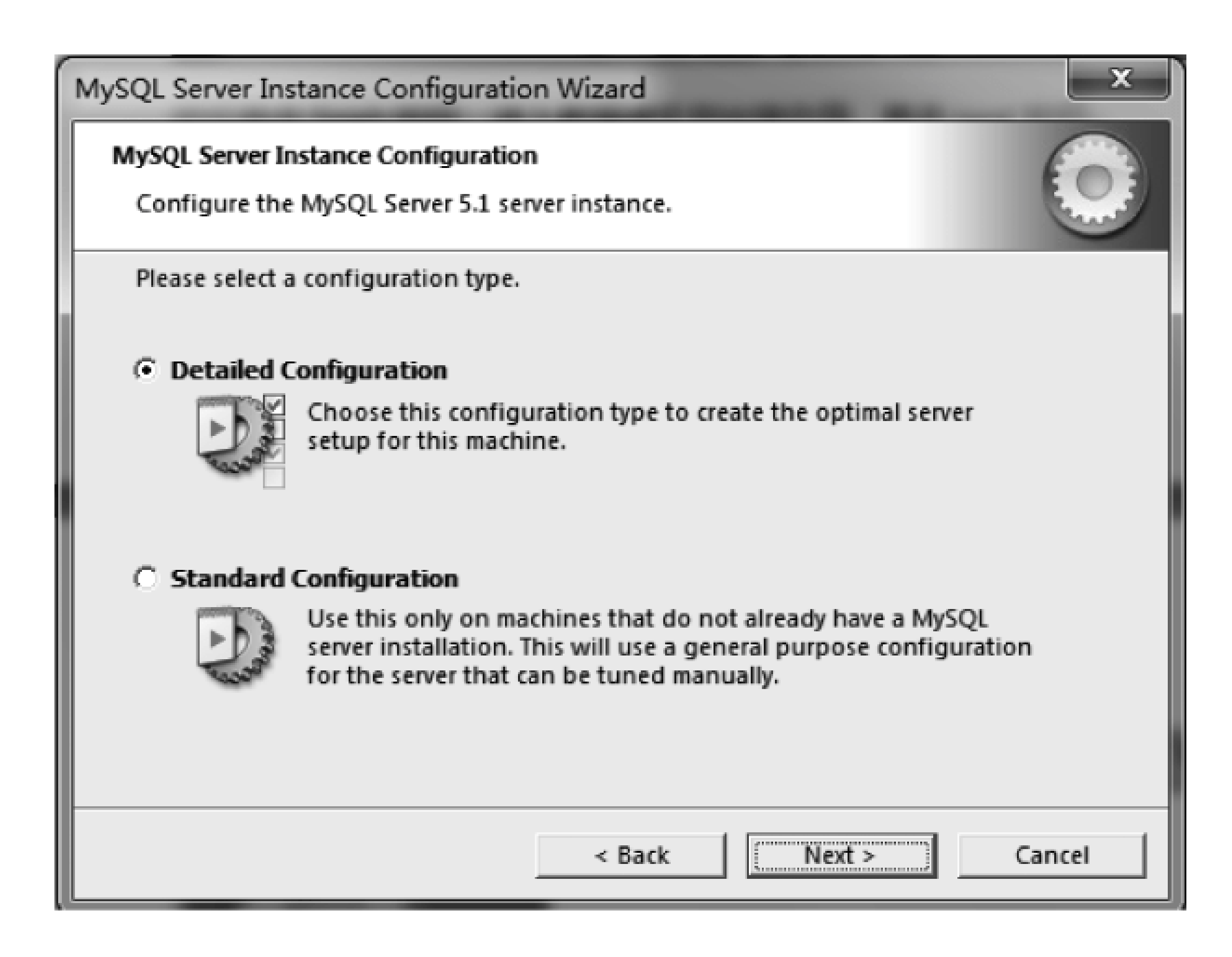


图 2.21　选择配置方式窗口



图 2.22　设置管理员账号和密码窗口

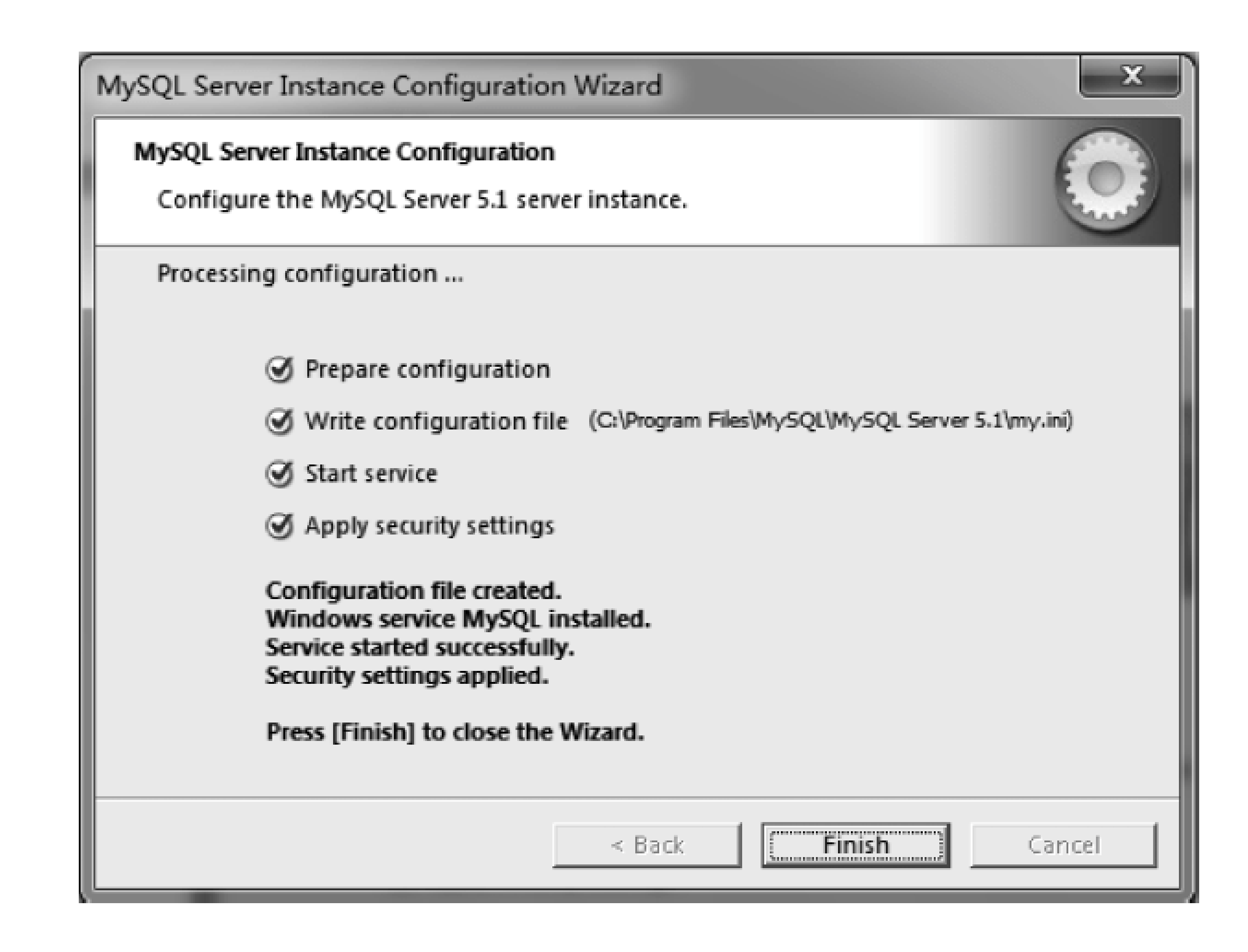


图 2.23　安装完成窗口

2.4 安装和配置 MyEclipse 8.0

MyEclipse 企业级工作平台(MyEclipse Enterprise Workbench,简称 MyEclipse)是对 Eclipse IDE 的扩展。利用它,我们可以在数据库和 Java EE 的开发、发布以及应用程序服务器的整合方面极大地提高工作效率。它是功能丰富的 Java EE 集成开发环境,包括了完备的编码、调试、测试和发布功能,完全支持 HTML、CSS、JavaScript、Struts、JSF、Hibernate、Spring 等。

2.4.1 安装 MyEclipse 8.0

MyEclipse 8.0 安装程序可以直接到网络上下载,安装也比较简单,所有的设置都直接默认即可完成安装。

2.4.2 配置 MyEclipse 8.0

(1) 单击 MyEclipse 菜单下的 Preferences 命令,弹出如图 2.24 所示的 Preferences (Filtered)对话框。

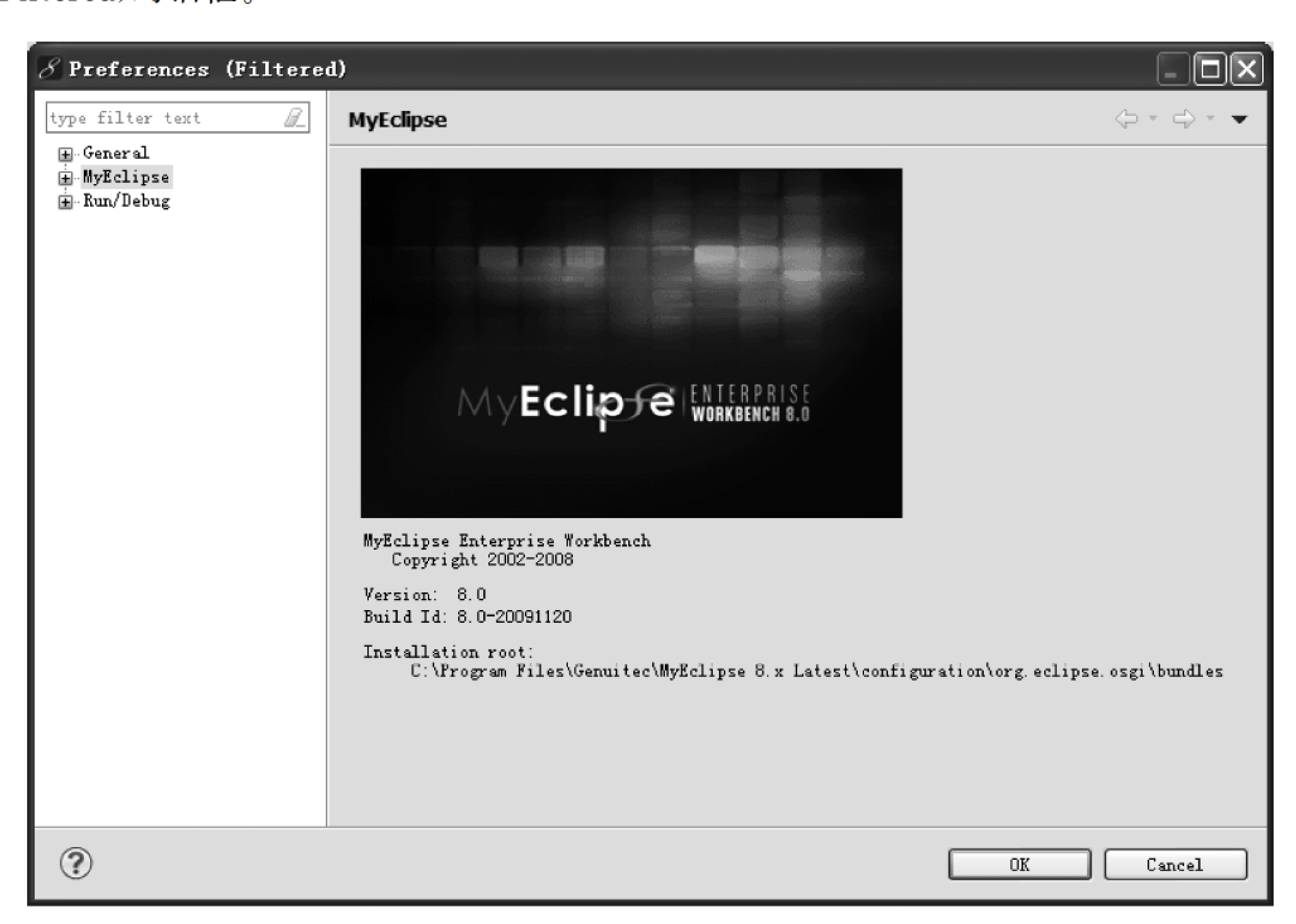


图 2.24 Preferences(Filtered)对话框

(2) 然后依次单击 MyEclipse|Servers|Tomcat,如图 2.25 所示。

(3) 选择 Tomcat 6.x,如图 2.26 所示。

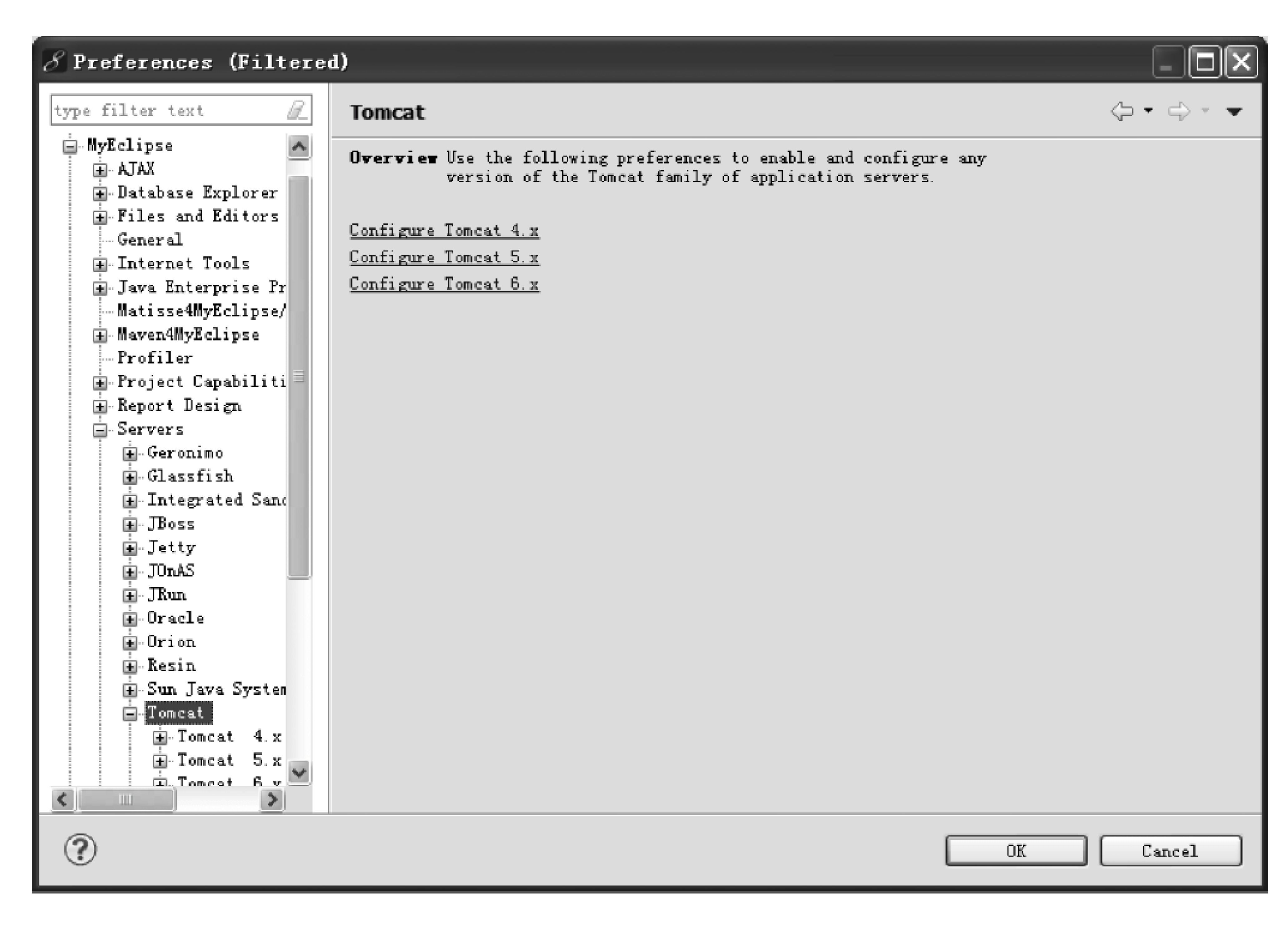


图 2.25　选择 Tomcat 配置项

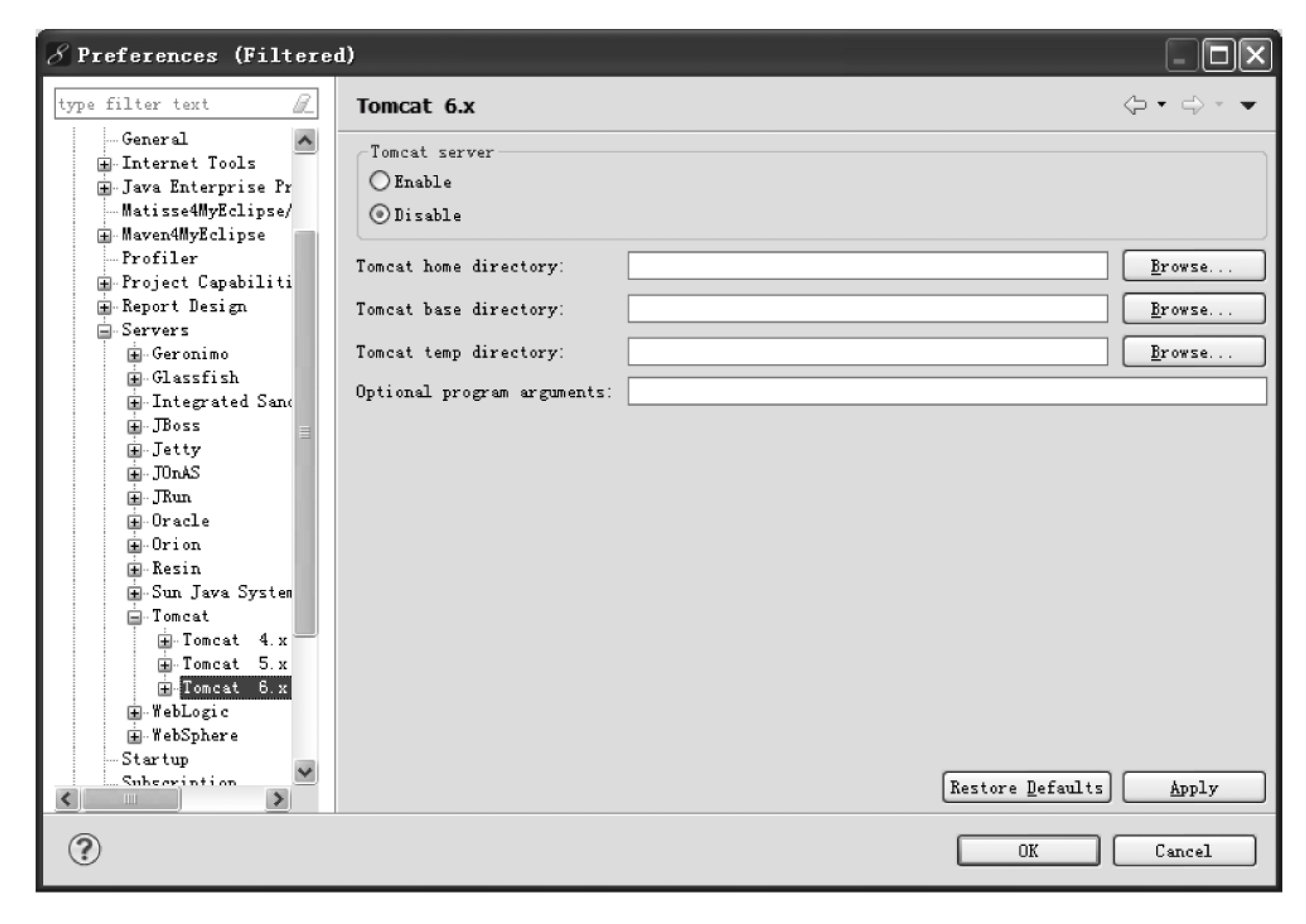


图 2.26　选择 Tomcat 6.x 配置项

（4）单击 Browse 按钮，选择准备好的 Tomcat 6.x 即可，并按照图 2.27 进行配置，最后依次单击 Apply 按钮和 OK 按钮，Tomcat 6.x 到此配置完毕。

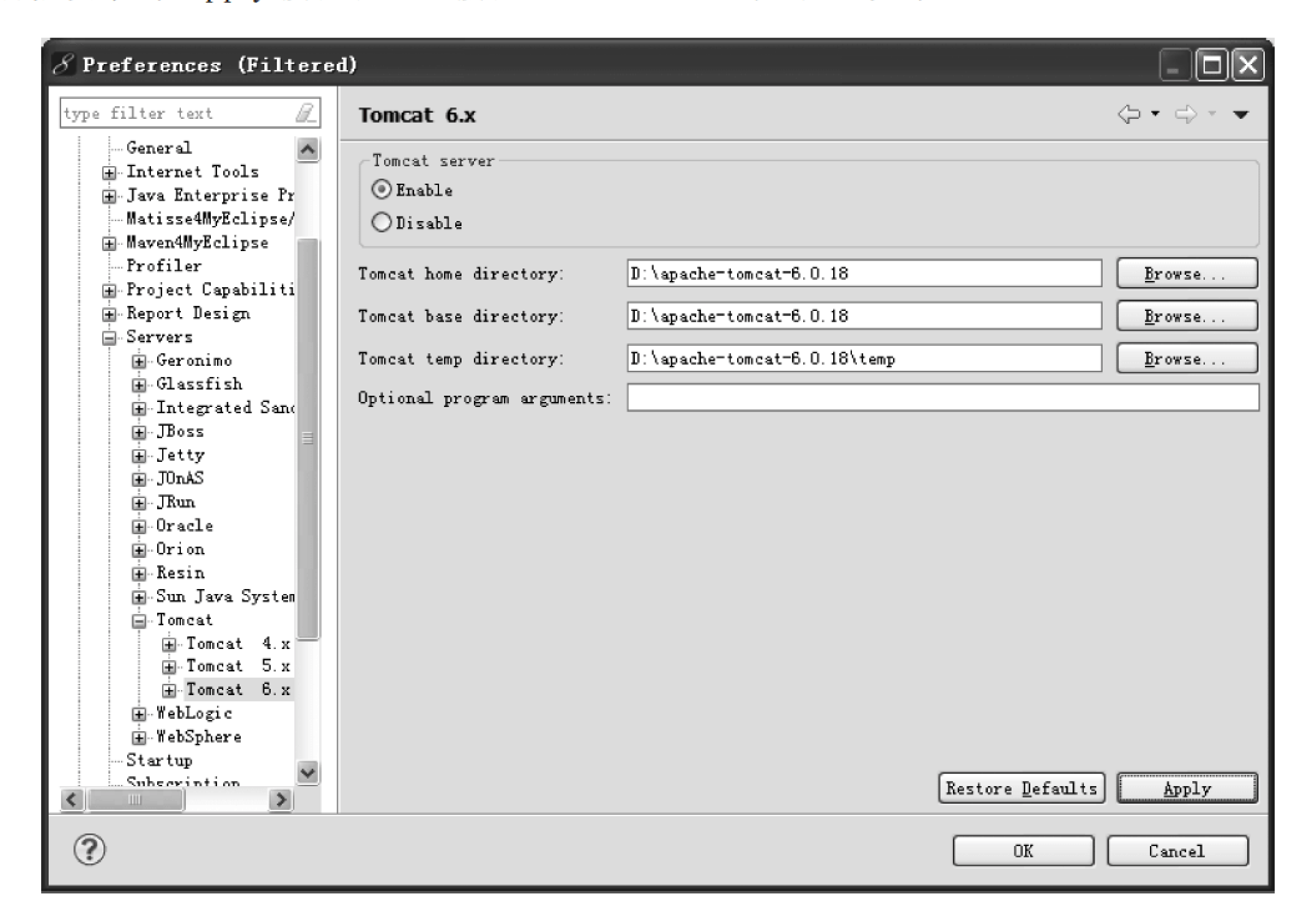


图 2.27　设置 Tomcat 6.x 参数

2.5　实践任务：在 MyEclipse 8.0 中建立并测试 Java Web 项目

1. 任务说明

通过本任务，我们将使用 MyEclipse 8.0 集成开发环境，创建第一个简单的 Java Web 项目，并进行发布测试。

2. 任务实施

（1）创建 Web 工程。启动 MyEclipse 8.0，在左侧 Workspace 的空白区域右击，在弹出的菜单中选择 New | Web Project 命令，弹出 New Web Project 对话框，如图 2.28 所示。

在对话框的 Project Name 里输入 Web 工程的名字：HelloWorld，并选择 Java EE 的版本号为 5.0，最后单击 Finish 按钮，完成工程的创建，如图 2.29 所示。

（2）发布 Web 工程。单击工具栏中的 Deploy 按钮，弹出如图 2.30 所示的 Project Deployments 对话框。

单击 Add 按钮，在弹出的对话框中，Server 项选择配置好的 Web 服务器 Tomcat 6.x。最后单击 Finish 按钮完成发布，如图 2.31 所示。

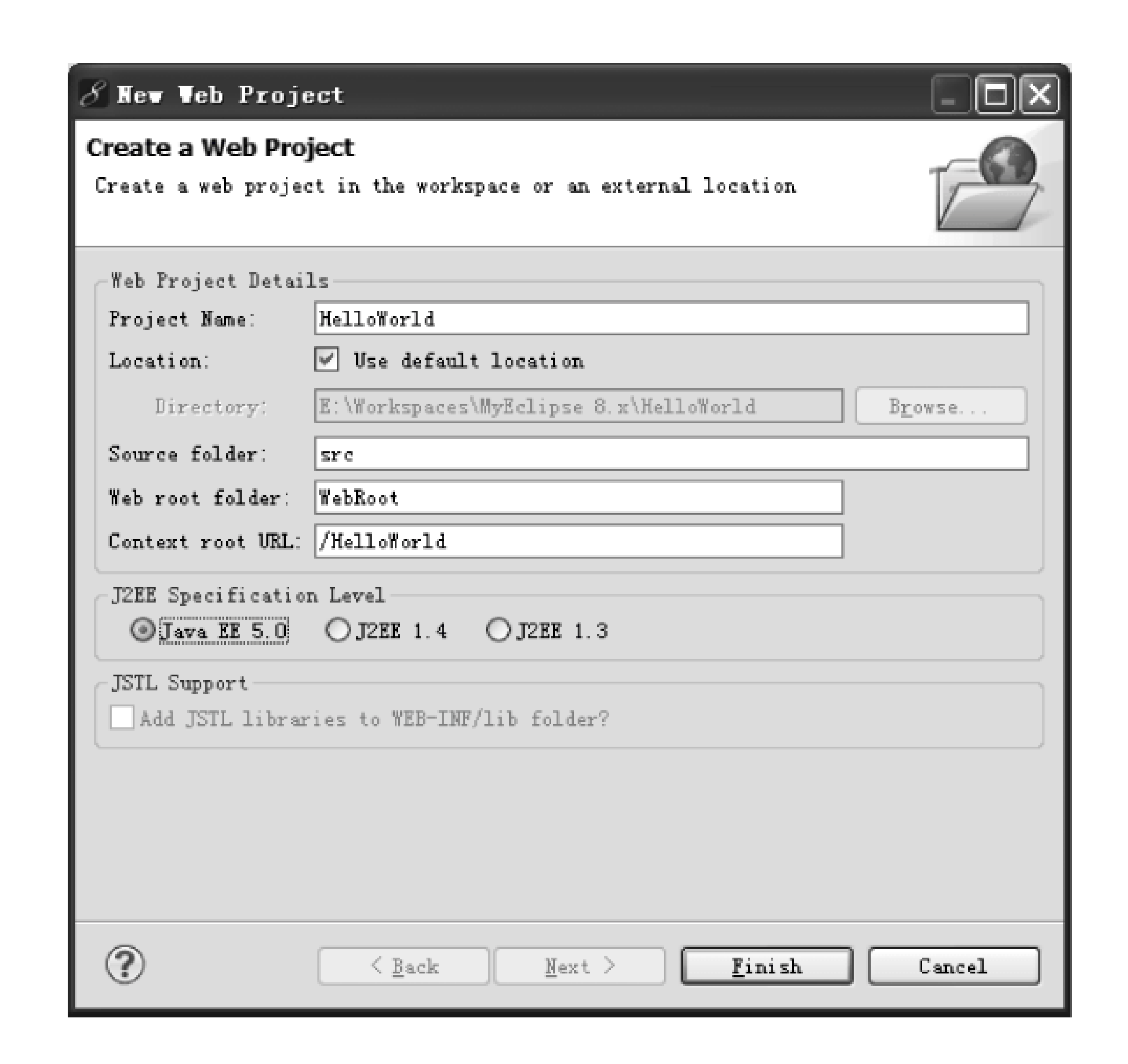


图 2.28　New Web Project 对话框

图 2.29　创建的 Web 工程

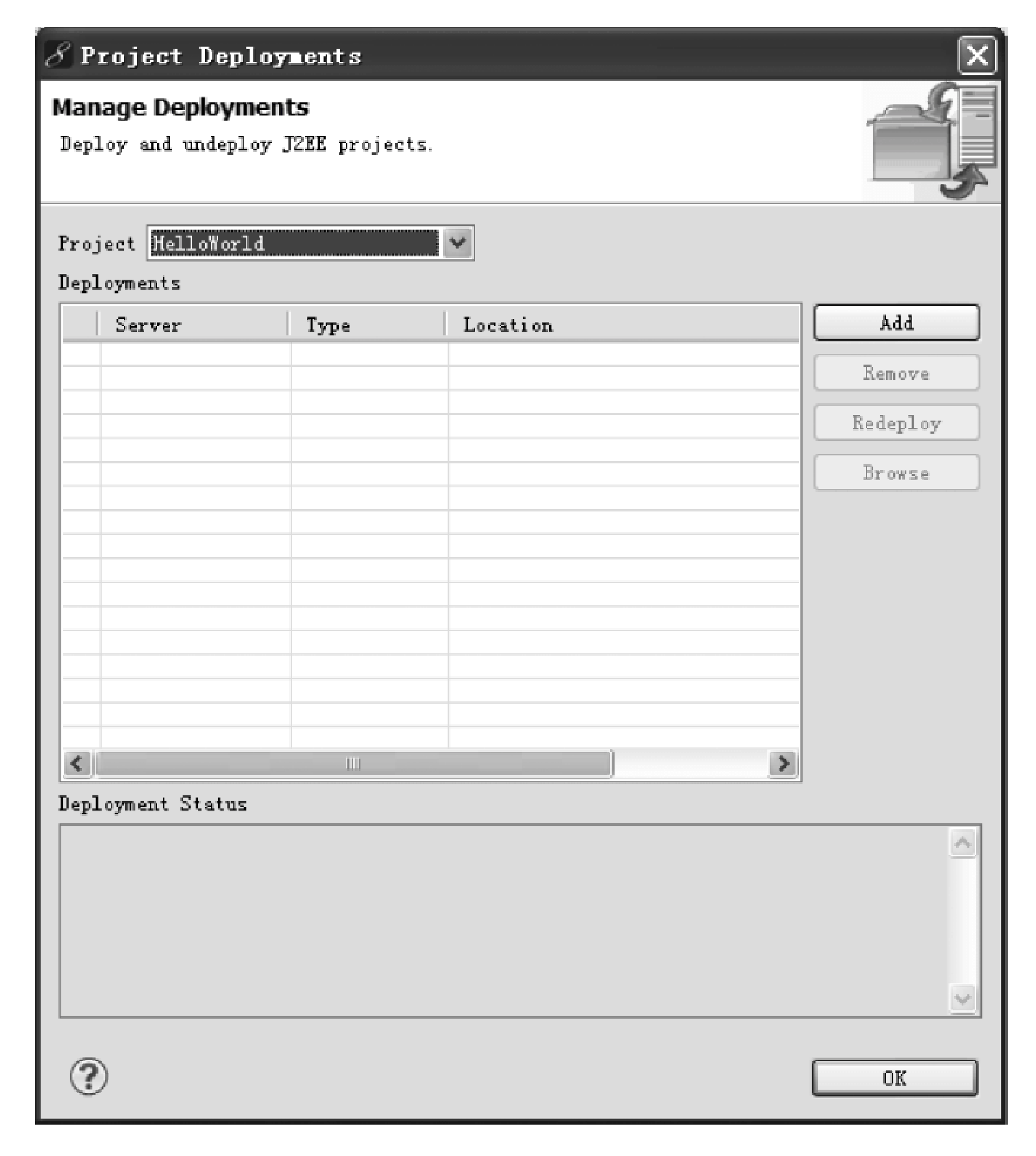


图 2.30　Project Deployments 对话框

(3) 启动 Tomcat 服务器。选择工具栏 Run|Tomcat 6.x|Start 选项，启动 Tomcat 服务器，控制台会出现启动服务器的过程信息，如图 2.32 所示。

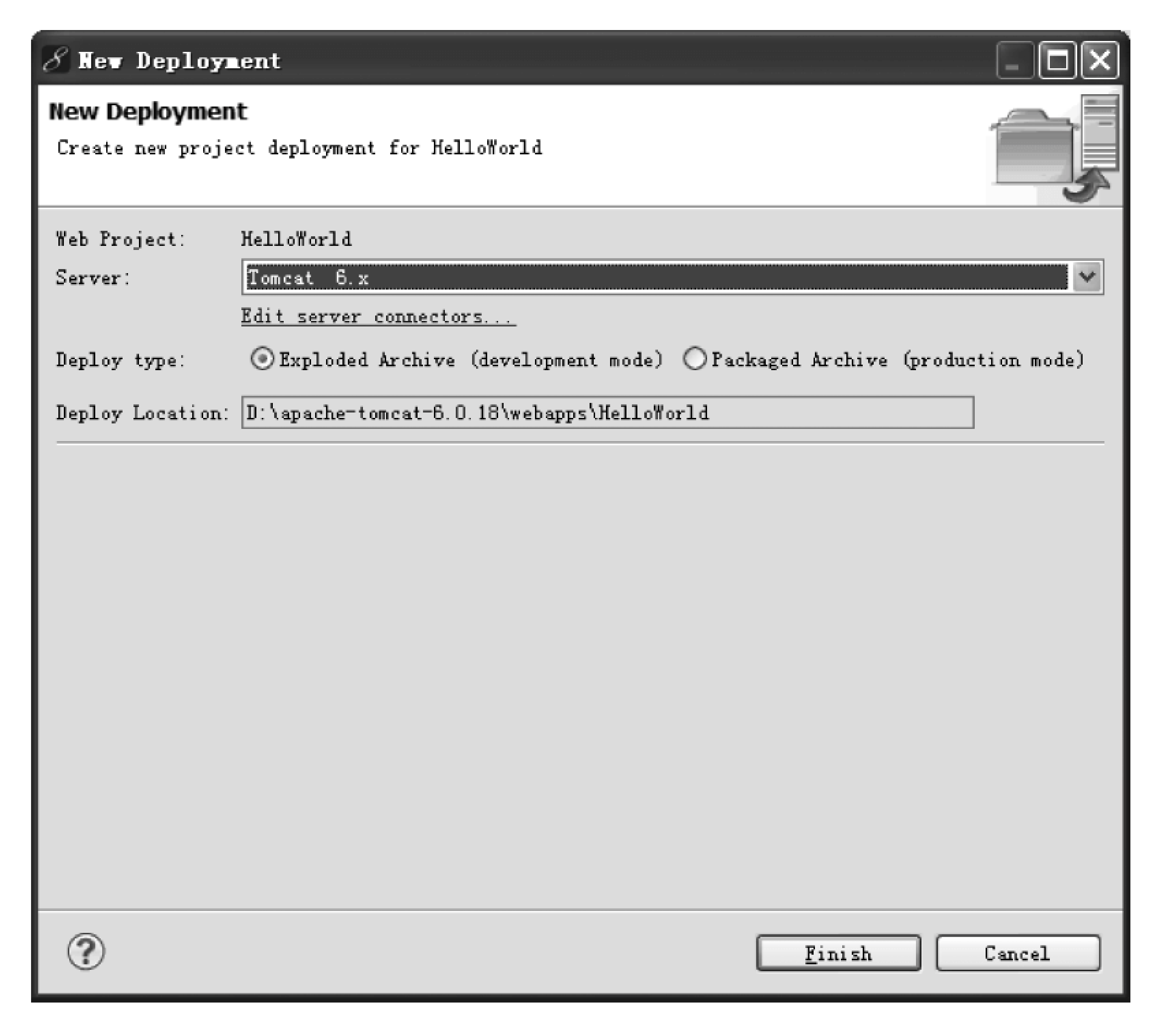


图 2.31 发布 Web 工程

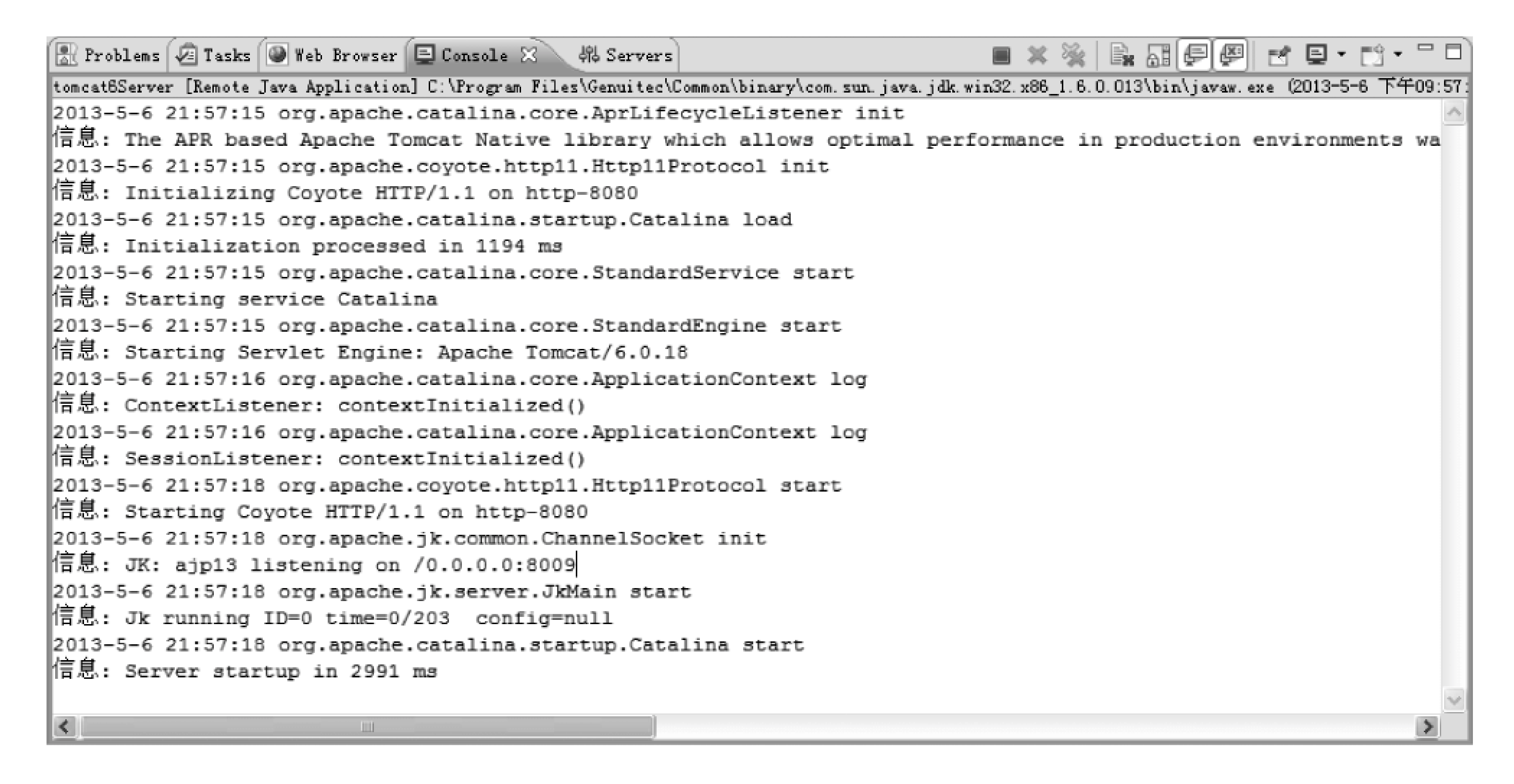


图 2.32 启动 Tomcat 服务器

（4）启动 IE，进行测试。单击工具栏中的 Open MyEclipse Web Browser 按钮，在地址栏里输入 http://localhost:8080/Helloworld/index.jsp，并按 Enter 键，出现如图 2.33 所示的图的界面，测试成功。

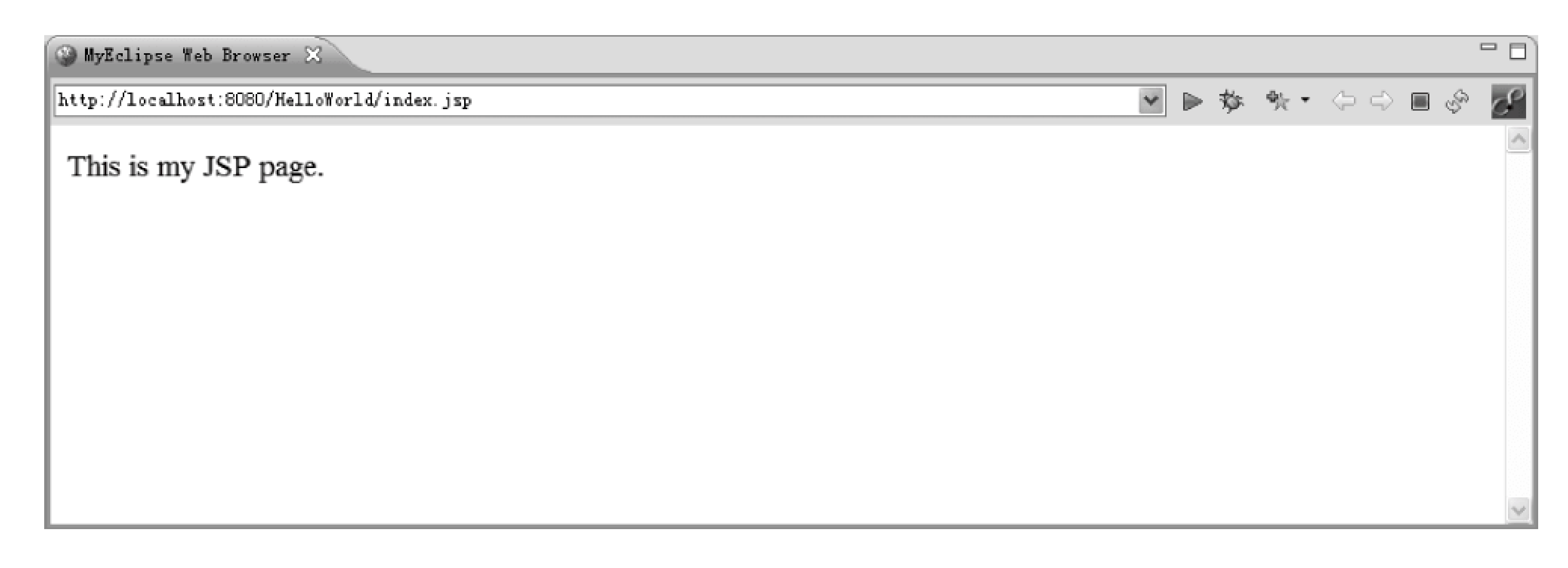


图 2.33　启动 IE,进行测试

3. 任务总结

本任务通过使用集成开发环境 MyEclipse 8.0 创建并发布测试了第一个比较简单的 Java Web 应用程序。

2.6　本章小结

本章对 Java Web 应用的开发运行环境的安装与配置进行了详细讲解,图文并茂,指导读者一步一步地操作,直到完成。最后通过一个简单的实践任务,完成了第一个 Java Web 应用的创建、发布和测试。

第3章

JSP 语法详解

在 JSP 文件里，主要由模板元素、指令元素、动作元素、脚本元素、声明、表达式、Scriptlets 和 JSP 内建对象组成。本章将介绍基本的 JSP 语法以及 JSP 页面中各元素的使用。

本章要点：

- JSP 页面的构成
- 模板元素
- 指令元素
- 脚本元素
- 动作元素

3.1 JSP 页面的构成

本节主要介绍 JSP 页面的基本结构，让读者对 JSP 页面有一个全面的了解。编写 JSP，可以使用编辑 HTML 的文本工具进行编辑，编辑完成后，把它保存为 *.jsp 文件即可。

JSP 代码一般情况下是由普通的 HTML 语句和特殊的嵌入标记组成。可以使用任何的编辑工具并按照常规方式来书写 HTML 语句，然后将动态部分用特殊的标记嵌入即可。这些标记通常以“<%”开始，并以“%>”结束。

1. JSP 简单实例

文件名为 web.jsp，源文件内容如下：

```
<html>
<body>
<%@ page language="java" contentType="text/html;charset=GBK"%>
<%@ page import="java.util.*"%>
<!--使用 java 语言编写-->
<%
    Date dnow = new Date();
    int dhours = dnow.getHours();
    int dminutes = dnow.getMinutes();
    int dseconds = dnow.getSeconds();
    out.print("服务器时间: "+dhours+":"+dminutes+":"+dseconds);
%>
<!--使用 JavaScript 脚本语言编写-->
```

```
<script language ="JavaScript">
    var dnow = new Date();
    dhours = dnow.getHours();
    dminutes = dnow.getMinutes();
    dseconds = dnow.getSeconds();
    document.write("<br>浏览器时间：" +dhours+":"+dminutes+":"+ dseconds);
</script>
</body>
</html>
```

运行结果如图 3.1 所示。

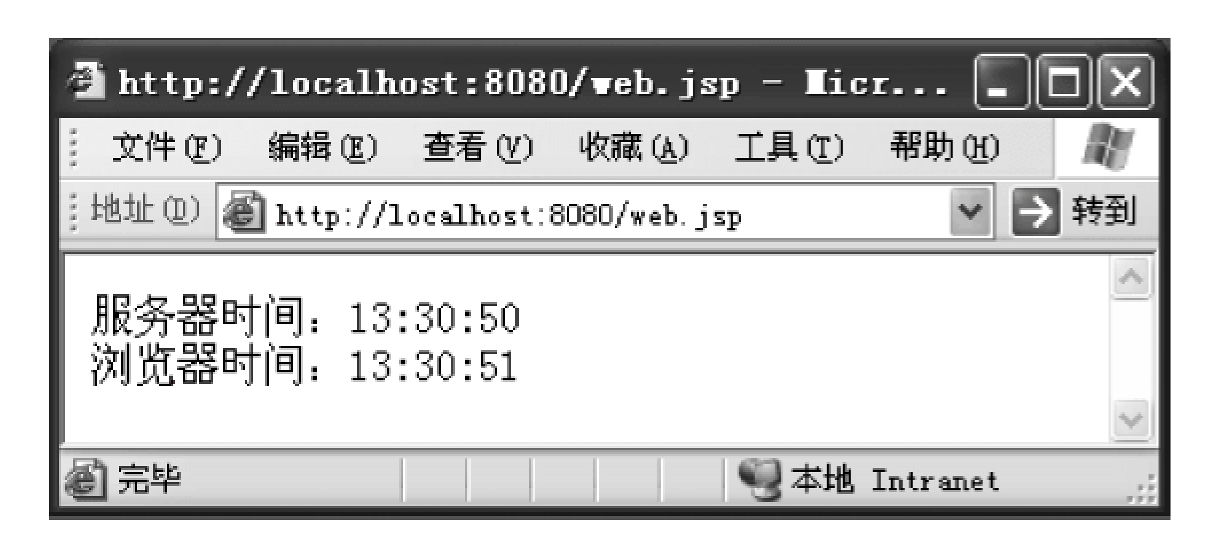


图 3.1　web.jsp 运行结果

在一个 JSP 页面中，可以由 5 类元素组成。

(1) 注释；

(2) HTML 或 MXL 内容；

(3) 脚本元素，主要有声明、表达式和 Scriptlet；

(4) 指令元素；

(5) 动作元素。

在这里我们主要介绍一下 JSP 中的注释。注释分为两种：一种称为显示注释(又称为 HTML 注释)；另一种称为隐式注释。

2. 显式注释

显示注释 JSP 语法如下：

```
<!--注释内容[<%=表达式%]-->
```

此种注释发给客户端。例如：

```
<html>
<body>
<%@ page language="java" contentType="text/html;charset=GBK"%>
<%@ page import="java.util.*"%>
 <!--获得服务器的时间：
   <%=new java.util.Date().toLocaleString()%>
 -->
</body>
</html>
```

程序运行的结果及在浏览器中打开的源文件如图 3.2 和图 3.3 所示。



图 3.2 date.jsp 运行结果

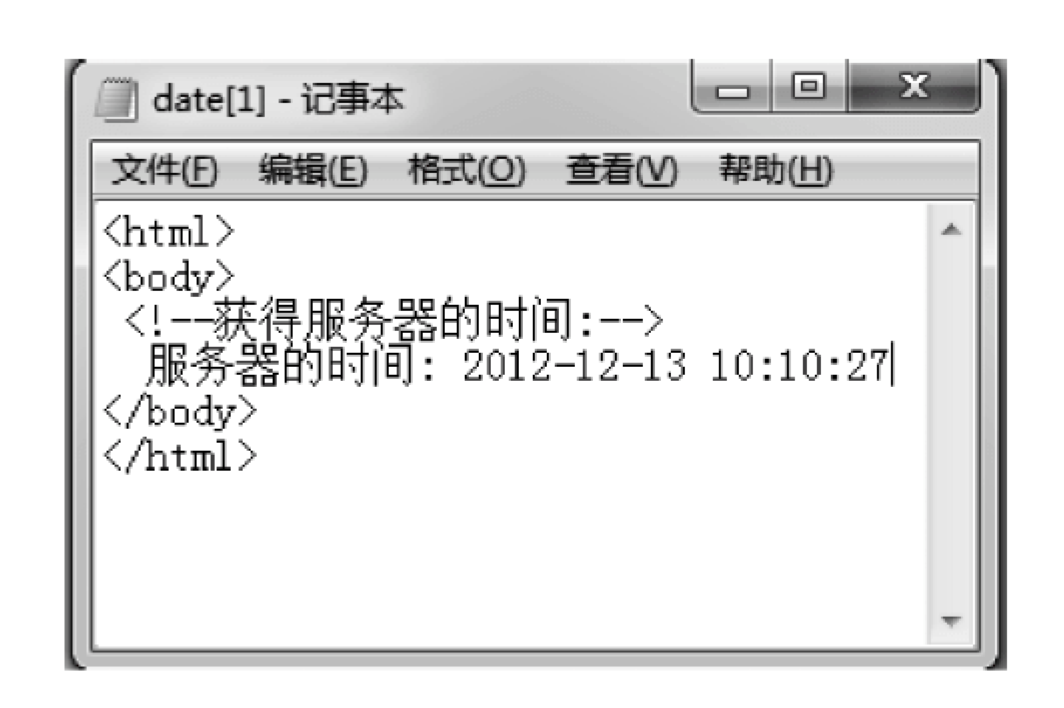


图 3.3 在浏览器中打开的源文件

3. 隐式注释

隐式注释 JSP 语法如下：

```
<%--注释内容--%>
```

此类注释写在 JSP 程序中，但不是发给客户。例如：

```
<html>
<body>
<%@ page language="java" contentType="text/html;charset=GBK"%>
<%@ page import="java.util.*"%>
 <%--获得服务器的时间--%>
 <%=new java.util.Date().toLocaleString()%>
</body>
</html>
```

程序运行结果及在浏览器中打开的源文件如图 3.4 和图 3.5 所示。

JSP 编译器不会对<%--...--%>之间的语句进行编译，<%--...--%>之间的内容也不会显示在客户的浏览器中，也不会在源代码中看到<%--...--%>标记，所以，可以在这里任意写注释语句。

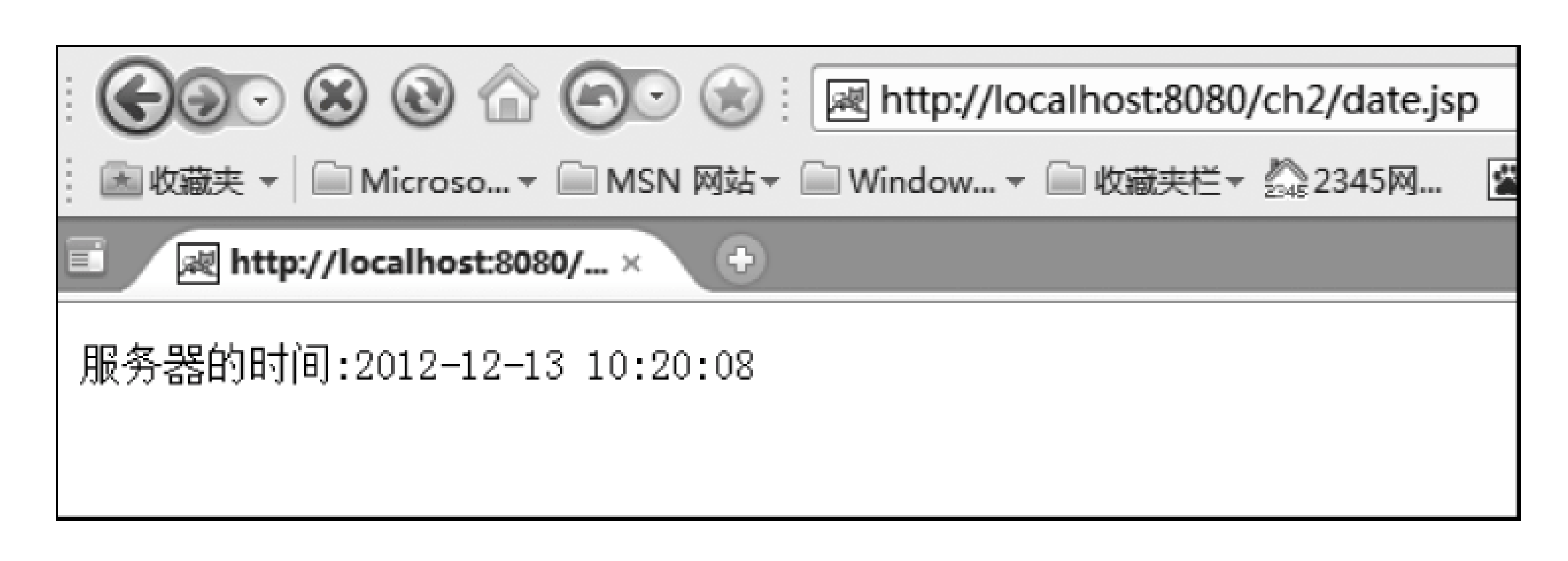


图 3.4　运行结果图



图 3.5　在浏览器中打开的源文件中看不到注释

3.2　指令元素

JSP 指令用于设置和整个 JSP 页面相关的属性，如页面的编码方法、包含文件以及是否为错误页面等。JSP 指令的用途非常简单，它只是告诉 JSP 引擎对 JSP 页面如何编译。

语法格式如下：

```
<%@ 指令名 属性="值" %>
```

说明：

<、%、@以及%、>之间不能有任何空格。

属性值两边的双引号可以替换为单引号。引号标记不能完全省略。

在属性中使用引号或\进行转义。

JSP 指令的内部就是一些指令和一连串的属性设置，如下所示。

```
<%@ 指令名 属性 1 = "value1" %>
<%@ 指令名 属性 2 = "value2" %>
```

也可以写成如下形式。

```
<%@ 指令名 属性 1 = "value1" 属性 2 = "value2" %>
```

JSP 指令最重要的两个指令是 page 指令和 include 指令。taglib 指令只有在用 JSP 标签的页面中使用。

3.2.1 page 指令

page 指令用来定义整个 JSP 页面的一些属性和这些属性的值。例如，我们可以用 page 指令定义 JSP 页面的 contentType 属性的值是"text/html;charset=GB2312"，这样，我们的页面就可以显示标准汉语。例如：

```
<%@ page contentType="text/html;charset=GB2312" %>
```

page 指令的格式如下：

```
<%@ page 属性 1= "属性 1 的值"  属性 2= "属性 2 的值"  … %>
```

属性值用双引号括起来，例如：

```
<%@ page contentType="text/html;charset=GB2312"  import="java.util.*" %>
```

如果为一个属性指定几个值话，这些值用逗号分割。page 指令只能给 import 属性指定多个值；其他属性只能指定一个值。例如：

```
<%@ page import="java.util.*,java.io.*,java.awt.*" %>
```

当你为 import 指定多个属性值时，JSP 引擎把 JSP 页面转译成的 JAVA 文件中会有如下的 import 语句。

```
import java.util.*;
import java.io.*;
import java.awt.*;
```

在一个 JSP 页面中，也可以使用多个 page 指令来指定属性及其值。需要注意的是：可以使用多个 page 指令给属性 import 几个值，但其他属性只能使用一次 page 指令指定该属性一个值。例如：

```
<%@ page contentType="text/html;charset=GB2312" %>
<%@ page import="java.util.*" %>
<%@ page import="java.awt.*" %>
```

注意，下列用法是错误的。

```
<%@ page contentType="text/html;charset=GB2312" %>
<%@ page contentType="text/html;charset=GB2312" %>
```

尽管指定的属性值相同，也不允许 2 次使用 page 给 contentType 属性指定属性值。

注：page 指令的作用对整个页面有效，与其书写的位置无关，但习惯把 page 指令写在 JSP 页面的最前面。

page 指令有如下属性。

(1) language 属性：定义 JSP 页面使用的脚本语言，该属性的值目前只能取"java"。为 language 属性指定值的格式如下：

```
<%@ page  language="java" %>
```

其中，language 属性的默认值是"java"，即如果你在 JSP 页面中没有使用 page 指令指定该属性的值，那么，JSP 页面默认有如下 page 指令。

```
<%@ page  language="java" %>
```

(2) import 属性：该属性的作用是为 JSP 页面引入 Java 核心包中的类，这样就可以在 JSP 页面的程序片部分、变量及函数声明部分、表达式部分使用包中的类。可以为该属性指定多个值，该属性的值可以是 Java 某包中的所有类或一个具体的类，例如：

```
<%@ page  import="java.io.*", "java.util.Date" %>
```

JSP 页面默认 import 属性已经有如下的值。

```
"java.lang.*"、"javax.servlet.*"、"javax.servlet.jsp.*"、"javax.servlet.http.*"
```

(3) contentType 属性：定义 JSP 页面响应的 MIME(Multipurpose Internet Mail Extension)类型和 JSP 页面字符的编码。属性值的一般形式是“MIME 类型”或“MIME 类型;charset=编码”。例如：

```
<%@ page contentType="text/html;charset=GB2312" %>
```

其中，contentType 属性的默认值是"text/html；charset=ISO-8859-1"。

(4) session 属性：用于设置是否需要使用内置的 session 对象。session 的属性值可以是 true 或 false，session 属性默认的属性值是 true。

(5) buffer 属性：内置输出流对象 out 负责将服务器的某些信息或运行结果发送到客户端显示，buffer 属性用来指定 out 设置的缓冲区的大小或不使用缓冲区。buffer 属性可以取值 none，设置 out 不使用缓冲区。buffer 属性的默认值是“8kb”。例如：

```
<%@ page buffer= "24kb" %>
```

(6) auotFlush 属性：指定 out 的缓冲区被填满时，缓冲区是否自动刷新。auotFlush 可以取值 true 或 false。auotFlush 属性的默认值是 true。当 auotFlush 属性取值 false 时，如果 out 的缓冲区填满时，就会出现缓存溢出异常。当 buffer 的值是 none 时，auotFlush 的值就不能设置成 false。

(7) isThreadSafe 属性：用来设置 JSP 页面是否可多线程访问。isThreadSafe 的属性值取 true 或 false。当 isThreadSafe 属性值设置为 true 时，JSP 页面能同时响应多个客户的请求；当 isThreadSafe 属性值设置成 false 时，JSP 页面同一时刻只能处理响应一个客户的请求，其他客户需排队等待。isThreadSafe 属性的默认值是 true。

(8) info 属性：该属性为 JSP 页面准备一个字符串，属性值是某个字符串。例如：

```
<%@ page info= "we are students" %>
```

可以在 JSP 页面中使用方法 getServletInfo()获取 info 属性的属性值。

注：当 JSP 页面被转译成 Java 文件时，转译成的类是 Servlet 的一个子类，所以在 JSP 页面中可以使用 Servlet 类的方法：getServletInfo()。

下面是 page 指令的简单演示实例，web.jsp 文件的内容如下：

```
<html>
<body>
<%@ page language="java" errorPage="error.jsp"
contentType="text/html;charset=GBK"%>
<%
    int dividend = 0;
    int divisor = 0;
    int result = 0;
    try {
        result=dividend/divisor;
    }
    catch(ArithmeticException zz) {
        throw new ArithmeticException("除数不能为零!");
    }
%>
</body>
</html>
```

error.jsp 文件的内容如下：

```
<html>
<body>
<%@ page language="java" isErrorPage="true"
contentType="text/html;charset=GBK"%>
当前页面是：error.jsp<br>
</body>
</html>
```

程序运行的结果如图 3.6 所示。



图 3.6 web.jsp 运行效果图

3.2.2 include 指令

如果需要在 JSP 页面内某处整体嵌入一个文件，就可以考虑使用这个指令标记。该指令标记语法如下：

```
<%@ include file= "文件的名字" %>
```

该指令标记的作用是在 JSP 页面出现该指令的位置处，静态插入一个文件。被插入的文件必须是可访问和可使用的，即该文件必须和当前 JSP 页面在同一 Web 服务目录中。所谓静态插入，就是当前 JSP 页面和插入的部分合并成一个新的 JSP 页面，然后 JSP 引擎再将这个新的 JSP 页面转译成 Java 类文件。因此，插入文件后，必须保证新合并成的 JSP 页面符合 JSP 语法规则，即能够成为一个 JSP 页面文件。比如，如果一个 JSP 页面使用 include 指令插入另一个 JSP 文件，被插入的这个 JSP 页面中有一个设置页面 contentType 属性的 page 指令。

```
<%@ page contentType="text/html;charset=GB2312" %>
```

而当前 JSP 页面已经使用 page 指令设置了 contentType 的属性值，那么新合并的 JSP 页面就出现了语法错误，当转译合并的 JSP 页面到 Java 文件时就会失败。

下面是 include 指令的演示实例。

文件 include.jsp 的内容如下：

```
<%@ page contentType="text/html; charset=gb2312"  %>
<%@ include file="head.jsp" %>
<%@ include file="body.htm" %>
<%@ include file="footer.jsp" %>
```

文件 head.jsp 的内容如下：

```
<html>
<body>
<table height=20 width=100% bgcolor=99ccff>
<tr>
<td align=center>==========header=========</td>
</tr>
</table>
</body>
</html>
```

文件 body.htm 的内容如下：

```
<html>
<body bgcolor=f45ef2>
<br>
<table height=20 width=100% bgcolor=9eecff>
<tr>
<td align=center>==========body=========</td>
</tr>
</table>
</body>
</html>
```

文件 footer.jsp 的内容如下：

```
<%@ page contentType="text/html; charset=gb2312"  %>
<br>footer...<br>
<table height=50 width=100% bgcolor=777777>
<tr>
<td align=center>
<hr>
{{{&copy;版权所有}}}<%=new java.util.Date()%>
</td>
</tr>
</table>
```

程序运行结果如图 3.7 所示。

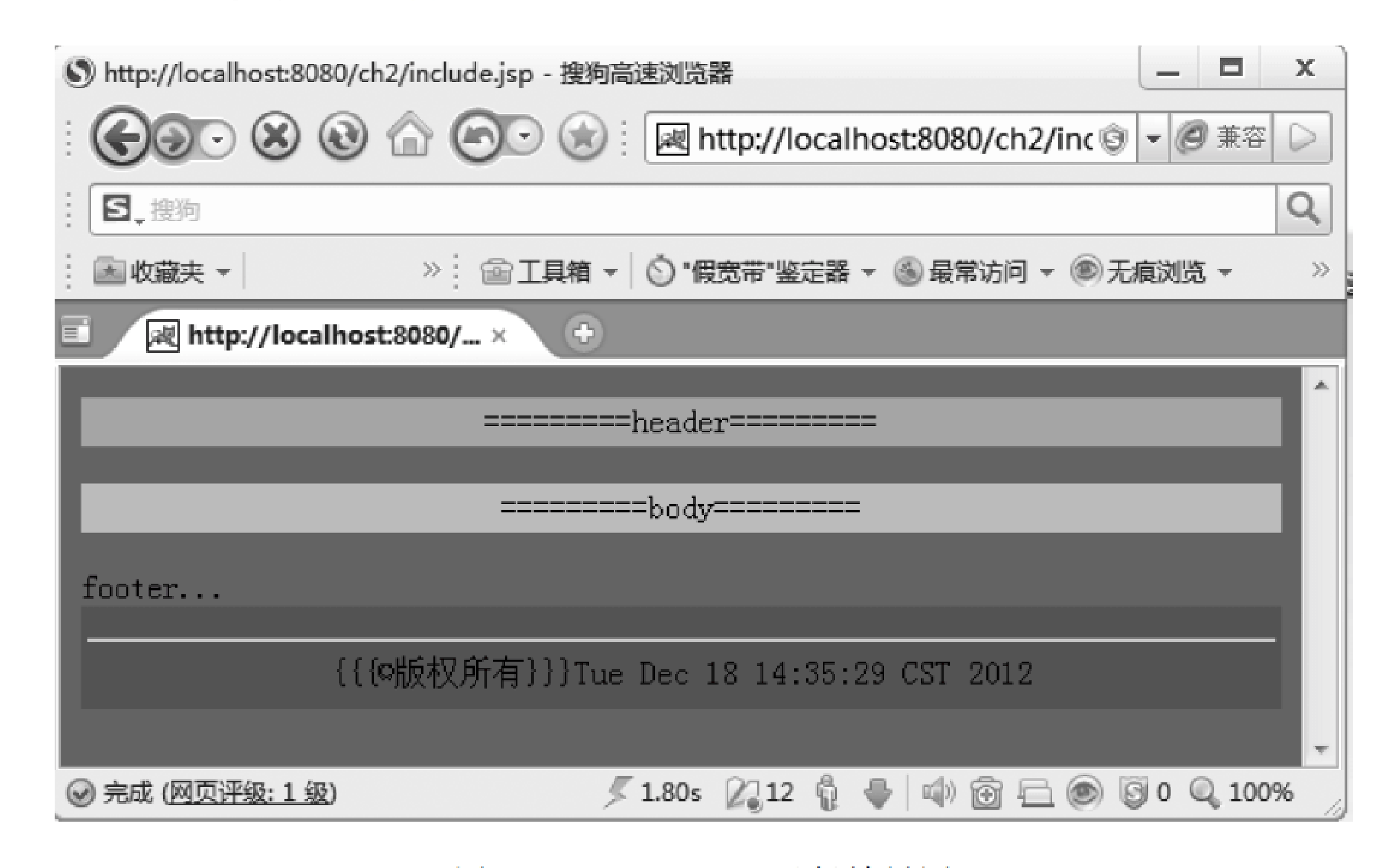


图 3.7 include.jsp 运行效果图

3.2.3 taglib 指令

taglib 指令定义一个标记库以及其自定义标记的前缀。

<% @ taglib %>指令声明此 JSP 文件使用了自定义的标记，同时引用标记库，也指定了它们的标记的前缀。

这里自定义的标记含有标记和元素之分。因为 JSP 文件能够转化为 XML，所以了解标记和元素之间的联系很重要。标记只不过是一个在意义上被抬高了点的标记，是 JSP 元素的一部分。

必须在使用自定义标记之前使用<% @ taglib %>指令，而且可以在一个页面中多次使用，但是前缀只能使用一次。

taglib 指令的 JSP 语法如下：

```
<%@ taglib uri="URIToTagLibrary" prefix="tagPrefix" %>
```

各属性的含义如下。

uri：一个 URI 标识标记库描述器。一个标记库描述器用来唯一地命名一组定制的标记，并且告诉包容器如何处理特殊的标记。

prefix：定义一个 prefix：tagname 形式的字符串前缀，用于定义定制的标记，所保留的前缀为 jsp、struts、jspx、java、servlet、sun、sunw。

3.3　脚本元素

JSP 脚本元素是 JSP 代码中使用最频繁的元素，特别是 Scriptlets，在早期的 JSP 代码编写中占有主导地位。可以通过 JSP 脚本元素在 JSP 中声明变量和方法。

3.3.1　声明

在 JSP 程序中，声明(Delaration)是一段 Java 代码，用来定义合法的变量和方法。声明后的变量和方法可以在 JSP 的任意位置使用。

JSP 语法如下：

```
<%! [变量]; [ 方法]+ ... %>
```

例如：

```
<html>
<body>
<%@ page language="java" contentType="text/html;charset=GBK"%>
<%@ page import="java.util. * " %>
<%! String getDate()
     {
         Date d=new Date();
         return d.toLocaleString();
     }
%>
服务器时间:
<%=getDate()%>
</body>
</html>
```

运行结果如图 3.8 所示。

3.3.2　表达式

表达式(Expression)元素表示的是一个在脚本语言中被定义的表达式，在运行后被自动转化为字符串，然后插入到这个表达式在 JSP 文件的位置显示。因为这个表达式的值已经被转化为字符串，所以能在一行文本中插入这个表达式。

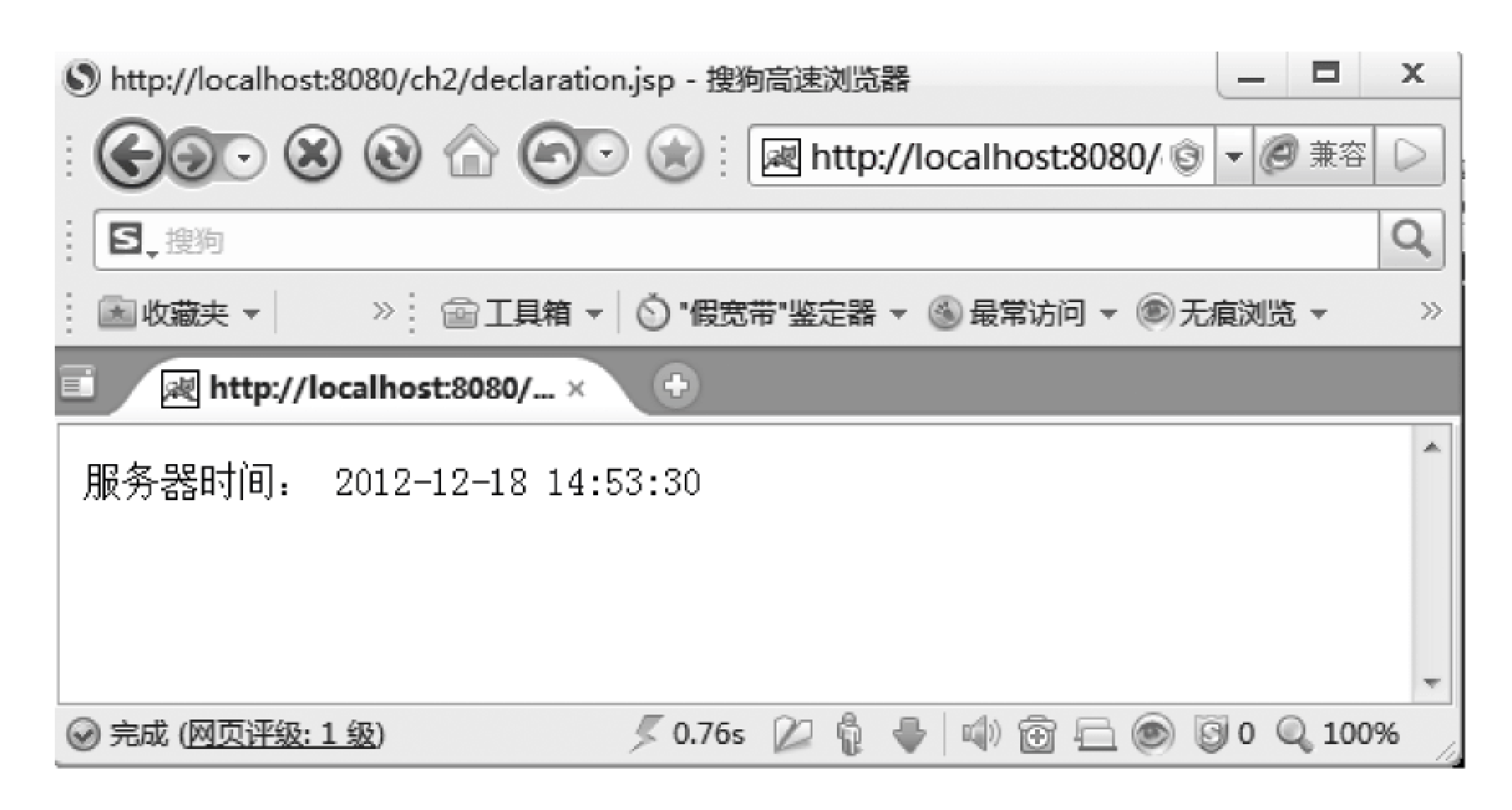


图 3.8　声明示例运行结果

表达式的 JSP 语法如下：

```
<%= expression %>
```

例如，3.3.1 小节中的<%=getDate()%>语句。

在 JSP 中使用表达式时，请记住以下两点。

(1) 不能用一个分号“;”来作为表达式的结束符。但是同样的表达式用在 Scriptlet 中就需要以分号来结尾了。

(2) 有时候表达式也能作为其他 JSP 元素的属性值。一个表达式能够变得很复杂，它可能由一个或多个表达式组成，这些表达式的顺序是从左到右。

例如：

```
<%@ page contentType="text/html;charset=GB2312" %>
<HTML>
<BODY  bgcolor=cyan><FONT size=1>
<P> Sin(0.9)除以 3 等于
  <%=Math.sin(0.90)/3%>
<P>3 的平方是：
  <%=Math.pow(3,2)%>
<P>12345679 乘 72 等于
  <%=12345679 * 72%>
<P> 5 的平方根等于
  <%=Math.sqrt(5)%>
<P>99 大于 100 吗？回答：
  <%=99>100%>
</BODY>
</HTML>
```

运行效果如图 3.9 所示。



图 3.9　bds.jsp 运行效果图

3.4　本章小结

本章主要介绍了 JSP 技术的基本语法，包括：注释、常用的指令、声明、表达式、Scriptlet 等。要熟练地使用以上基本语法，为进一步的学习打好基础。

第4章 JSP 内置对象详解

有些对象不用声明就可以在 JSP 页面的脚本部分使用，这就是 JSP 的内置对象。JSP 根据 Servlet API 而提供了某些隐含对象。可以使用标准的变量来访问这些对象，并且不用编写任何额外的代码，就可以在 JSP 中自动使用它。在 JSP 页面中可以获得的主要 9 个隐含对象变量如下。

（1）out 对象：其功能是把信息回送到客户端的浏览器中。

（2）response 对象：其功能是处理服务器端对客户端的一些响应。

（3）request 对象：其功能是用来获得客户端的信息。

（4）application 对象：用来保存网站的一些全局变量。

（5）session 对象：用来保存单个用户访问时的一些信息。

（6）cookie 对象：将服务器端的一些信息写到客户端的浏览器中。

（7）pageContext 对象：提供了访问和放置页面中共享数据的方式。

（8）page 对象：JSP 网页本身对象。

（9）exception 对象：针对错误网页，未捕捉的例外。

本章要点：

- out 对象
- request 对象
- response 对象
- application 对象
- page 对象
- exception 对象

4.1 out 对象

out 对象是 javax.servlet.jsp.JspWriter 类的一个子类的对象，它的作用是把信息回送到客户端的浏览器中。在 out 对象中，最常用的方法就是 print()和 println()。在使用 print()或 println()方法时，由于客户端是浏览器，因此向客户端输出时，可以使用 HTML 中的一些标记，如 out.println("<h1>Hello，JSP</h1>")。

out 对象提供的常用方法如下所述。

（1）print()：把 Java 对象原始数据类型输入客户端的缓冲区。

(2) println():把内容输出到客户端,还在后面添加一个空行,但是被浏览器忽略。

(3) out.write():功能和 out.print 相同。

(4) out.newLine():功能是输出一个换行符。

(5) out.flush():功能是输出缓冲的内容。

(6) out.close():功能是关闭输出流。

out 对象的生命周期是当前页面。因此对于每一个 JSP 页面,都有一个 out 对象。

【例 4.1】 out 对象的演示实例,文件 out.jsp 的内容如下:

```
<%@ page contentType="text/html; charset=gb2312" language="java" %>
<%
    response.setContentType("text/html");
    out.println("学习使用 out 对象: <br><hr>");
    out.println("<br>out.println(boolean):");
    out.println(true);
    out.println("<br>out.println(char):");
    out.println('a');
    out.println("<br>out.println(char[]):");
    out.println(new char[]{'a','b'});
    out.println("<br>out.println(double):");
    out.println(2.3d);
    out.println("<br>out.println(float):");
    out.println(43.2f);
    out.println("<br>out.println(int):");
    out.println(34);
    out.println("<br>out.println(long):");
    out.println(2342342343242354L);
    out.println("<br>out.println(object):");
    out.println(new java.util.Date());
    out.println("<br>out.println(string):");
    out.println("string");
    out.println("<br>out.newLine():");
    out.newLine();
    out.println("<br>out.getBufferSize():");
    out.println(out.getBufferSize());
    out.println("<br>out.getRemaining():");
    out.println(out.getRemaining());
    out.println("<br>out.isAutoFlush():");
    out.println(out.isAutoFlush());
    out.flush();
    out.println("<br>调用 out.flush()后,测试是否输出");
    out.close();
    out.println("<br>调用 out.close()后,测试是否输出");
    out.clear();
    out.println("<br>调用 out.clear()后,测试是否输出");
%>
```

运行结果如图 4.1 所示。



图 4.1　out 对象演示实例

下面的例子是使用 out 对象向客户输出包括表格等内容的信息。

```
<%@ page contentType="text/html;charset=GB2312" %>
<%@ page import="java.util.*" %>
<HTML>
<BODY>
  <% int a=100;long b=300;boolean c=true;
     out.println("<H1>这是标题 1 字体的大小</HT1>");
     out.println("<H2>这是标题 2 字体的大小</HT2>");
     out.print("<BR>");
     out.println(a);   out.println(b);   out.println(c);
  %>
  <Center>
  <p><Font size=2 >以下是一个表格</Font>
  <%out.print("<Font face=隶书 size=2 >");
      out.println("<Table Border >");
      out.println("<TR >");
      out.println("<TH width=80>"+"姓名"+"</TH>");
      out.println("<TH width=60>"+"性别"+"</TH>");
      out.println("<TH width=200>"+"出生日期"+"</TH>");
      out.println("</TR>");
      out.println("<TR >");
      out.println("<TD >"+"刘甲一"+"</TD>");
      out.println("<TD >"+"男"+"</TD>");
      out.println("<TD >"+"1978 年 5 月"+"</TD>");
      out.println("</TR>");
      out.println("<TR>");
      out.println("<TD >"+"林 霞"+"</TD>");
```

```
            out.println("<TD >"+"女"+"</TD>");
            out.println("<TD >"+"1979 年 8 月"+"</TD>");
            out.println("<TD width=100>"+"这是表格"+"</TD>");
            out.println("</TR>");
            out.println("</Table>");
            out.print("</Font>")    ;
    %>
    </Center>
  </BODY>
  </HTML>
```

运行结果如图 4.2 所示。



图 4.2　用 out 对象输出各种信息

4.2　request 对象

HTTP 通信协议是客户与服务器之间一种提交(请求)信息与响应信息(Request/Respone)的通信协议。在 JSP 中,内置对象 request 封装了用户提交的信息,那么该对象调用相应的方法可以获取封装的信息,即使用该对象可以获取用户提交的信息。

客户通常使用 HTML 表单向服务器的某个 JSP 页面提交信息,表单的一般格式如下:

```
<FORM   method= get | post   action= "提交信息的目的地页面">
    提交手段
</FORM>
```

其中,<FORM>是表单标签;method 取值 get 或 post。get 方法和 post 方法的主要区别是:使用 get 方法提交的信息会在提交的过程中显示在浏览器的地址栏中,而 post 方法提交的信息不会显示在地址栏中。提交手段包括:通过文本框、列表、文本区等,例如:

```
<FORM action="tom.jsp" method= "post" >
     <INPUT type="text" name="girl" value= "yes" >
     <INPUT TYPE="submit" value="送出" name= "submit">
```

```
</FORM>
```

该表单使用 post 方法向页面 tom.jsp 提交信息，提交信息的手段是：在文本框输入信息，其中默认信息是 yes；然后单击“送出”按钮向服务器的 JSP 页面 tom.jsp 提交信息。

在 tom.jsp 页面使用 request 对象的 getParameter(String s)方法获取该表单通过 text 提交的信息，例如：

```
request.getParameter("girl");
```

1. request 对象的常用方法

request 对象的常用方法如下。

(1) getAttribute(String name)：返回 name 指定的属性值，如果指定的值不存在，则返回一个 null 值。

(2) getAttributeNames()：返回 request 对象的所有属性的名称，结果为一个枚举类的实例。

(3) getCookies()：返回客户端的 Cookie 对象，结果是一个 Cookie 数组。

(4) getParameter(String name)：获得客户端传送给服务器端的参数值，该参数由 name 指定。

(5) getParameterNames()：返回客户端传送给服务器端的所有参数的名称，其结果也是一个枚举类的实例。

(6) getParameterValues(String name)：获得指定参数的所有值，参数由 name 指定。

(7) getRequestURI()：获取发出请求字符串的客户端的地址。

(8) getRemoteAddr()：获得客户端的 IP 地址。

(9) getRemoteHost()：获取客户端的名称。

2. 利用 request 对象获得 Form 表单的信息

通常用得最多的就是客户端请求的参数名称和参数值信息。得到某参数值的语法如下：

```
request.getParameter("param1")
```

可以通过 request 对象的 getParameterNames()方法得到客户端传递过来的所有参数的值。例如：

```
Enumeration params = request.getParameterNames();
```

下面是 HTML 表单的演示实例。

【例 4.2】 读取表单数据。

login.htm 的内容如下：

```
<html>
<body>
<form action="login.jsp" method="post">
<p>姓名：<input type="text"   size="20"   name="username"></p>
<p>密码：<input type="password"   size="20" name="password"></p>
<p><input type="submit" value="提 交"> </p>
```

```
</form>
</body>
</html>
```

login. jsp 的内容如下：

```
<%@ page contentType="text/html;charset=GBK" %>
<%
     request.setCharacterEncoding("GBK");
     String strUserName = "";
     String strUserPWD = "";
     strUserName = request.getParameter("username");
     strUserPWD = request.getParameter("password");
%>
姓名：<%=username%><br>
密码：<%=password%>
```

使用 request 对象获取用户提交数据的所有参数名称和值实例。

对 login. jsp 的内容修改如下：

```
<%@ page contentType="text/html;charset=GBK" import="java.util.*"%>
<%
     String current_param = "";
     request.setCharacterEncoding("GBK");
     Enumeration params = request.getParameterNames();
     while(params.hasMoreElements()) {
     current_param = (String)params.nextElement();
     out.print("变量名:" + current_param + "<br>");
     out.print("变量值:"+request.getParameter(current_param)+"<br>");
}%>
```

在对应的文本框中输入 Jack 和 12345678 后，运行结果如图 4.3 所示。

单击“提交”按钮后，运行结果如图 4.4 所示。

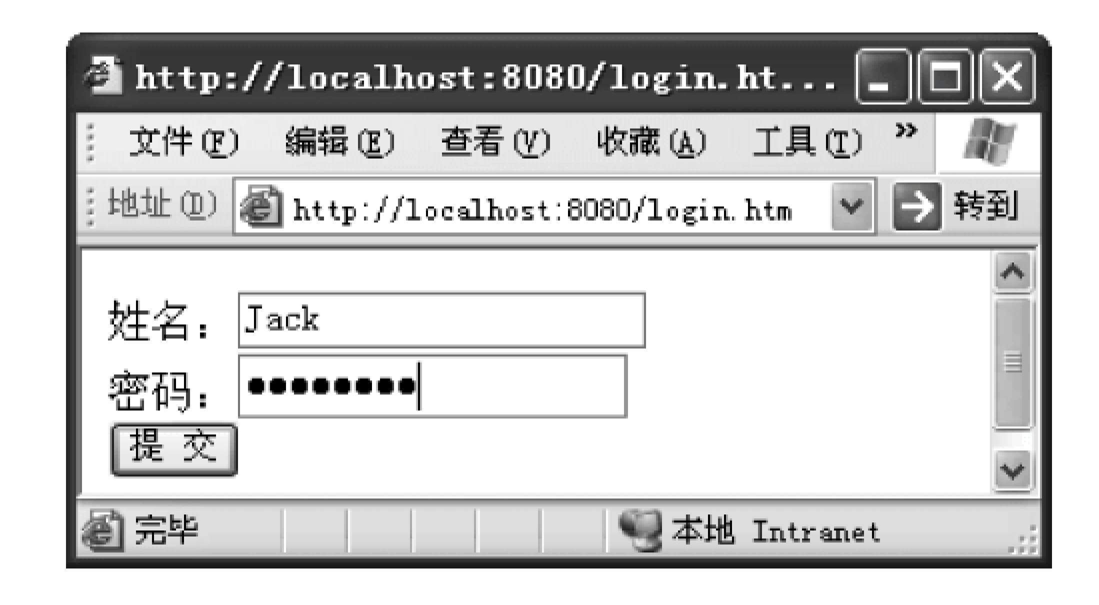



图 4.3 login. htm 页面运行结果



图 4.4 提交后的运行结果

3. 利用 request 对象得到客户的信息

利用 request 对象可以获得客户提交的信息。

【例 4.3】 获取 JSP 文件目录、客户端的地址以及服务器的端口等。

程序名称为 request.jsp，代码如下：

```
<%@ page contentType="text/html;charset=GBK" %>
<html>
<body>
<%
    out.println("<br>获取使用的协议:");
    out.println(request.getProtocol());
    out.println("<br>获取客户端的 IP 地址:");
    out.println(request.getRemoteAddr());
    out.println("<br>获取客户端的名称:");
    out.println(request.getRemoteHost());
    out.println("<br>获取客户端的用户名称:");
    out.println(request.getRemoteUser());
    out.println("<br>获取服务器的名称:");
    out.println(request.getServerName());
    out.println("<br>获取服务器使用的端口号");
    out.println(request.getServerPort());
%>
</body>
</html>
```

运行结果如图 4.5 所示。



图 4.5 request.jsp 运行结果



汉字问题两种解决方法

当利用 request.getParameter 得到 Form 种元素的时候，默认的情况字符编码为 ISO-8859-1，这种编码不能正确地显示汉字。

目前有两种解决方法。

(1) 在执行操作之前，设置 request 的编码格式，语法如下：

```
request.setCharacterEncoding("GBK");
```

(2) 转换字符编码的代码如下：

```
<%String str=request.getParameter("boy");
```

```
        byte  b[]=str.getBytes("ISO-8859-1");
        str=new String(b);
%>
```

4.3　response 对象

当客户访问一个服务器的页面时，会提交一个 HTTP 请求，服务器收到请求时，返回 HTTP 响应。响应和请求类似，也有某种结构，每个响应都由状态行开始，可以包含几个头及可能的信息体（网页的结果输出部分）。

用 request 对象获取客户请求提交的信息，与 request 对象相对应的对象是 response 对象。我们可以用 response 对象对客户的请求作出动态响应，向客户端发送数据。比如，当一个客户请求访问一个 JSP 页面时，该页面用 page 指令设置页面的 contentType 属性的值是 text/html，那么 JSP 引擎将按照这种属性值响应客户对页面的请求，将页面的静态部分返回给客户。如果想动态地改变 contentType 的属性值，就需要用 response 对象改变页面的这个属性的值，作出动态的响应。

response 对象是一个 javax.servlet.http.HttpServletResponse 类的子类的对象，它封装了 JSP 产生的响应，然后被发送到客户端以响应客户的请求，用于向客户端发送数据。

对于 response 对象，最常用到的是 sendRedirect()方法，可以使用这个方法将当前客户端的请求转到其他页面去。相应的代码格式如下：

```
response.sendRedirect("URL 地址");
```

【例 4.4】　response 对象 sendRedirect()方法的演示实例。

文件 login.jsp 的内容如下：

```
<%@ page contentType="text/html; charset=gb2312" language="java" %>
<html>
<body>
<form method=post action="checklogin.jsp">
<table>
<tr>
<td>请输入你的姓名:</td>
<td><input type=text name=name></td>
</tr>
<tr><td>请输入你的密码:</td>
<td><input type=password name=password></td>
</tr>
<tr>
<td><input type=submit value=登录></td>
</tr>
</table>
</body>
</html>
```

文件 checklogin.jsp 的内容如下：

```
<%@ page contentType="text/html; charset=gb2312" %>
<html>
<body>
<%--进行登录检查--%>
<%
   String name=request.getParameter("name");
   String password=request.getParameter("password");
   if(name.equals("Jack"))
     response.sendRedirect("sucess.jsp?name="+name);
else
     response.sendRedirect("login.jsp");
%>
</body>
</html>
```

文件 sucess.jsp 的内容如下：

```
<%@ page contentType="text/html; charset=gb2312" language="java" %>
<%
     String name=getParameter("name");
     out.print(name+"登录成功");
%>
```

在对应的文本框中输入 Jack 和 12345678 后，如图 4.6 所示。



图 4.6　login.jsp 运行结果

单击“登录”按钮后，运行如图 4.7 所示。



图 4.7　登录后的运行结果

可以利用 JSP 动态改变客户端的响应，使用的语法如下：

```
response.setHeader(String name,String value);
```

例如，可以让客户端每隔 1 秒自动刷新一次。

【例 4.5】　response 对象 setHeader()方法演示实例。

```
<%@ page contentType="text/html;charset=GBK" %>
<%@ page import="java.util.*" %>
<p>现在的时间是：<br>
<%
out.println(new Date().toLocaleString());
response.setHeader("Refresh","1");
%>
```

运行结果如图 4.8 所示。



图 4.8　setHeader()方法运行结果

4.4　session 对象

HTTP 协议是一种无状态协议。一个客户向服务器发出请求(Request)，然后服务器返回响应(Respons)，连接就被关闭了。在服务器端不保留连接的有关信息，因此当下一次连接时，服务器已没有以前的连接信息了，无法判断这一次连接和以前的连接是否属于同一客户。因此，必须使用会话记录有关连接的信息。

从一个客户打开浏览器连接到服务器，到客户关闭浏览器离开这个服务器，称作一个会话。当一个客户访问一个服务器时，可能会在这个服务器的几个页面反复连接、反复刷新一个页面或不断地向一个页面提交信息等，服务器应当通过某种办法知道这是同一个客户，这就需要 session(会话)对象。

session 对象是 java.servlet.http.HttpSession 类的子类的对象，它表示当前的用户会话信息。在 session 中保存对象，在当前用户连接的所有页面中都是可以被访问到的。

可以使用 session 对象存储用户登录网站时候的信息。当用户在页面之间跳转时，存储在 session 对象中的变量不会被清除。

session 的信息保存在容器里，session 的 ID 保存在客户机的 Cookie 中。一般情况下，用户首次登录系统时容器会给此用户分配一个唯一标识 sessionID，这个 ID 用于区分其他的用户，当用户退去系统时，这个 session 就会自动消失。

session 对象提供的主要方法如下。

(1) getAttribute(String name)：获取与指定名称 name 相联系的信息。

(2) getAttributeNames()：返回 session 对象中存储的每一个属性对象，其结果为一

个枚举类的实例。

(3) getID():返回一个唯一的标识,这些标识为每个 session 而产生。

(4) getLastAccessedTim():返回当前 session 对象最后被客户发送的时间。

(5) invalidate():销毁这个 session 对象,使得和它绑定的对象都失效。

(6) setAttribute(String name,Object value):设置指定名称 name 的属性值 value,并将之存储在 session 对象中。

【例 4.6】 session 对象的演示实例。控制不经过密码验证不能在浏览器中直接后续网页。

文件 login.htm 的内容如下:

```
<html>
<body>
<form action="login.jsp" method="post">
<p>姓名:<input type="text"  size="20"  name="username"></p>
<p>密码:<input type="password"  size="20" name="password"></p>
<p><input type="submit" value="提 交"> </p>
</form>
</body>
</html>
```

文件 login.jsp 的内容如下:

```
<%@ page contentType="text/html;charset=GBK" %>
<%
    request.setCharacterEncoding("GBK");
    String strUserName = "";
    String strUserPWD = "";
    strUserName = request.getParameter("username");
    strUserPWD = request.getParameter("password");
    session.setAttribute("username ",strUserName);
    session.setAttribute("password",strUserPWD);
    if(strUserName.equals("Jack") && strUserPWD.equals("123456")
        response.sendRedirect("main.jsp");
    else
        response.sendRedirect("login.htm");
%>
```

文件 main.jsp 的内容如下:

```
<%@ page contentType="text/html;charset=GBK" %>
<%
    request.setCharacterEncoding("GBK");
    if(session.getAttribute("username")==null &&
      session.getAttribute("password")==null)
      response.sendRedirect("login.htm");
    else
      out.println("进入主页");
%>
```

如果在浏览器的地址栏直接输入 http://localhost:8080/main.jsp，不是进入 main.jsp 页面，而是直接跳到 login.htm 页面。

4.5　实践任务 1：使用 session 对象存储顾客的姓名和购买的商品

1. 任务说明

本任务模拟简单的购物流程，首先在 Example3_17.jsp 页面中输入姓名，如图 4.9 所示。然后送出进入 first.jsp 页面，在此页面输入你想购买的商品，如图 4.10 所示。单击“送出”按钮后进入 account.jsp 页面，在此页面显示出顾客的姓名和购买的商品信息，如图 4.11 所示。



图 4.9　Example3_17.jsp 页面运行结果



图 4.10　first.jsp 页面运行结果



图 4.11　account.jsp 页面运行结果

2. 任务实施

(1) 编写 Example3_17.jsp 页面。

```
<%@ page contentType="text/html;charset=GB2312" %>
<HTML>
<BODY bgcolor=cyan><FONT size=1>
    <% session.setAttribute("customer","顾客");%>
<P>输入你的姓名连接到第一百货: first.jsp
<FORM action="first.jsp" method=post name=form>
    <INPUT type="text" name="boy">
    <INPUT TYPE="submit" value="送出" name=submit>
</FORM>
</FONT>
</BODY>
</HTML>
```

(2) 编写 first.jsp 页面。

```
<%@ page contentType="text/html;charset=GB2312" %>
<HTML>
<BODY bgcolor=cyan><FONT Size=1>
```

```
    <% String s=request.getParameter("boy");
       session.setAttribute("name",s);
    %>
<P>这里是第一百货
<P>输入你想购买的商品连接到结账: account.jsp
<FORM action="account.jsp" method=post name=form>
    <INPUT type="text" name="buy">
    <INPUT TYPE="submit" value="送出" name=submit>
</FORM>
</FONT>
</BODY>
</HTML>
```

(3) 编写 account.jsp 页面。

```
<%@ page contentType="text/html;charset=GB2312" %>
<%! //处理字符串的方法:
    public String getString(String s)
    { if(s==null){
          s="";
      }
      try {byte b[]=s.getBytes("ISO-8859-1");
              s=new String(b);
      }
       catch(Exception e){   }
       return s;
    }
%>
<HTML>
<BODY bgcolor=cyan><FONT Size=1>
    <% String s=request.getParameter("buy");
       session.setAttribute("goods",s);
    %>
<BR>
<% String 顾客=(String)session.getAttribute("customer");
     String 姓名=(String)session.getAttribute("name");
     String 商品=(String)session.getAttribute("goods");
     姓名=getString(姓名);
     商品=getString(商品);
%>
<P>这里是结账处
<P><%=顾客%>的姓名是: <%=姓名%>
<P>您选择购买的商品是: <%=商品%>
</FONT>
</BODY>
</HTML>
```

(4) 发布、测试。

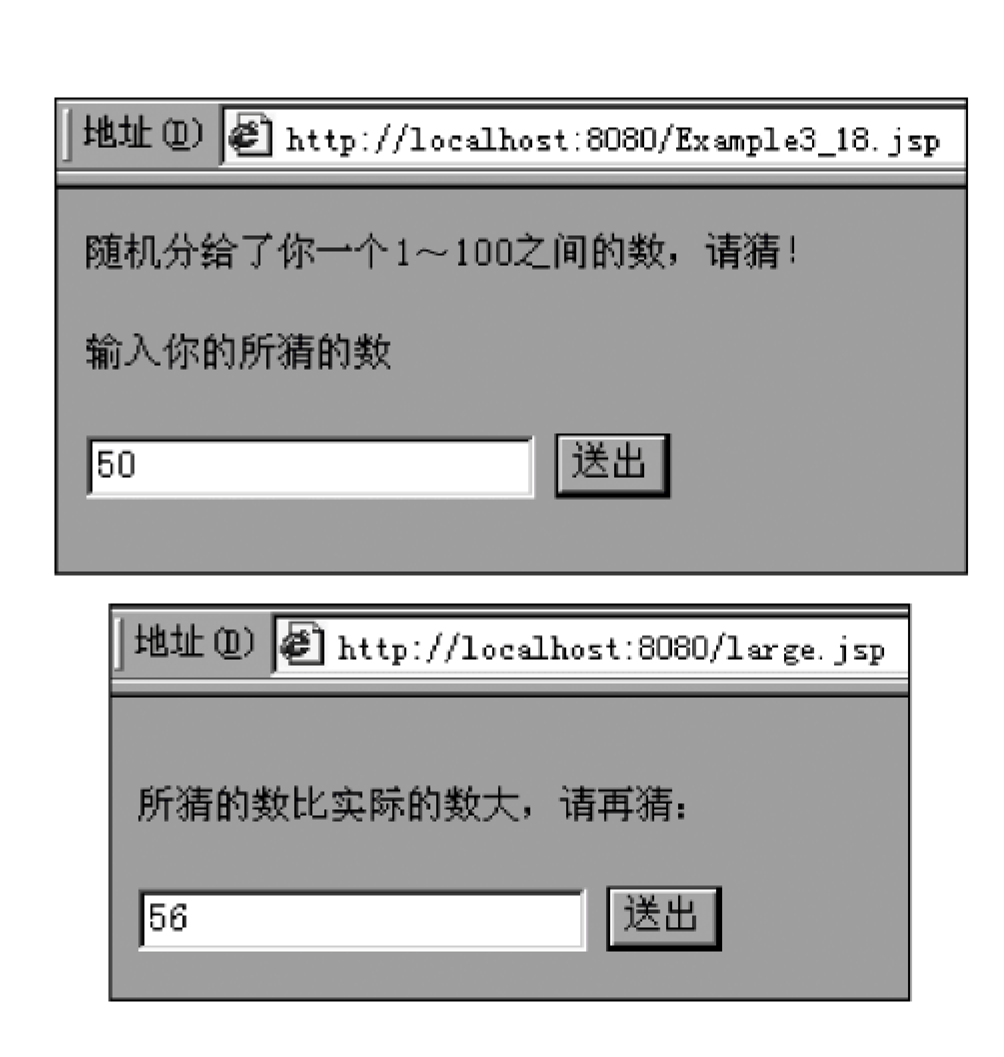
3. 任务总结

通过本任务,测试了可以使用 session 对象存储用户登录网站时候的信息。当用户在页面之间跳转时,存储在 session 对象中的变量不会被清除。

4.6　实践任务 2:猜数字的小游戏

1. 任务说明

当客户访问服务器上的 Example3_18.jsp 时,随机分配给客户一个 1～100 之间的数,然后将这个数字存在客户的 session 对象中。客户在表单里输入一个数,来猜测分配给自己的那个数字。客户输入一个数字后,提交给 result.jsp,该页面负责判断这个数是否和客户的 session 对象中存放的那个数字相同。如果相同就连接到 success.jsp;如果不相同就连接到 large.jsp 或 small.jsp。然后,客户在这些页面再重新提交数字到 result 页面,如图 4.12 所示。



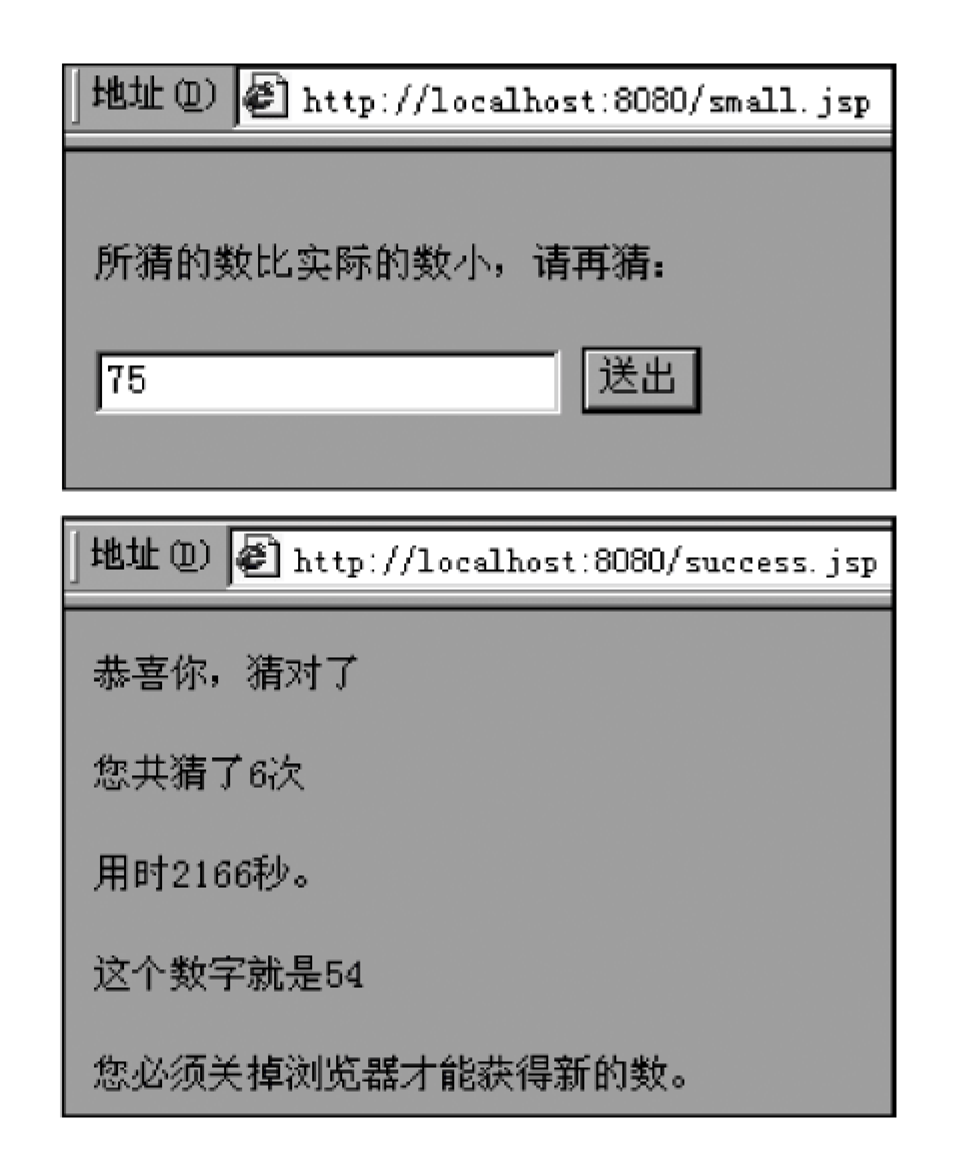


图 4.12　猜数字小游戏运行结果

2. 任务实施

(1) 编写 Example3_18.jsp 页面。

```
<%@ page contentType="text/html;charset=GB2312" %>
<HTML>
<BODY bgcolor=cyan><FONT Size=1>
<P>随机分给了你一个 1～100 之间的数,请猜!
<%
     int number=(int)(Math.random() * 100)+1;
     session.setAttribute("count",new Integer(0));
     session.setAttribute("save",new Integer(number));
%>
```

```
<BR>
<P>输入你的所猜的数
<FORM action="result.jsp" method="post" name=form>
    <INPUT type="text" name="boy" >
    <INPUT TYPE="submit" value="送出" name="submit">
</FORM>
</FONT>
</BODY>
</HTML>
```

(2) 编写 result.jsp 页面。

```
<%@ page contentType="text/html;charset=GB2312" %>
<HTML>
<BODY bgcolor=cyan><FONT Size=1>
<BR>
<% String str=request.getParameter("boy");
     if(str==null) {
        str="0";
     }
     int guessNumber=Integer.parseInt(str);
     Integer integer=(Integer)session.getAttribute("save");
     int realnumber=integer.intValue();
     if(guessNumber==realnumber) {
        int n=((Integer)session.getAttribute("count")).intValue();
        n=n+1;
        session.setAttribute("count",new Integer(n));
        response.sendRedirect("success.jsp");
     }
     else if(guessNumber>realnumber) {
        int n=((Integer)session.getAttribute("count")).intValue();
        n=n+1;
        session.setAttribute("count",new Integer(n));
        response.sendRedirect("large.jsp");
     }
     else if(guessNumber<realnumber) {
        int n=((Integer)session.getAttribute("count")).intValue();
        n=n+1;
        session.setAttribute("count",new Integer(n));

        response.sendRedirect("small.jsp");
     }
%>
</FONT>
</BODY>
</HTML>
```

(3) 编写 large.jsp 页面。

```
<%@ page contentType="text/html;charset=GB2312" %>
<HTML>
<BODY bgcolor=cyan><FONT Size=1>
<BR>
<P>所猜的数比实际的数大,请再猜:
<FORM action="result.jsp" method="get" name=form >
    <INPUT type="text" name="boy" >
    <INPUT TYPE="submit" value="送出" name="submit">
</FORM>
</FONT>
</BODY>
</HTML>
```

(4) 编写 small.jsp 页面。

```
<%@ page contentType="text/html;charset=GB2312" %>
<HTML>
<BODY bgcolor=cyan><FONT Size=1>
<BR>
<P>所猜的数比实际的数小,请再猜:
<FORM action="result.jsp" method="post" name=form>
    <INPUT type="text" name="boy" >
    <INPUT TYPE="submit" value="送出" name="submit">
</FORM>
</FONT>
</BODY>
</HTML>
```

(5) 编写 success.jsp 页面。

```
<%@ page contentType="text/html;charset=GB2312" %>
<HTML>
<BODY bgcolor=cyan><FONT Size=1>
<%   int count=((Integer)session.getAttribute("count")).intValue();
     int num=((Integer)session.getAttribute("save")).intValue();
     long startTime=session.getCreationTime();
     long endTime=session.getLastAccessedTime();
%>
<P>恭喜你,猜对了
<BR>
<P>您共猜了<%=count%>次
<P>用时<%=(endTime-startTime)/1000%>秒
<P>这个数字就是<%=num%>
<P>您必须关掉浏览器才能获得新的数
</FONT>
</BODY>
</HTML>
```

(6) 发布、测试。

3. 任务总结

session 对象是 java.servlet.http.HttpSession 类的子类的对象，它表示当前的用户会话信息。在 session 中保存对象，在当前用户连接的所有页面中都是可以被访问到的。

4.7 application 对象

当一个客户第一次访问服务器上的一个 JSP 页面时，JSP 引擎创建一个和该客户相对应的 session 对象，当客户在所访问的网站的各个页面之间浏览时，这个 session 对象都是同一个，直到客户关闭浏览器，这个 session 对象才被取消；而且不同客户的 session 对象是互不相同的。与 session 对象不同的是 application 对象。服务器启动后，就产生了这个 application 对象。当一个客户访问服务器上的一个 JSP 页面时，JSP 引擎为该客户分配这个 application 对象，当客户在所访问的网站的各个页面之间浏览时，这个 application 对象都是同一个，直到服务器关闭，这个 application 对象才被取消。与 session 对象不同的是，所有客户的 application 对象是相同的一个，即所有的客户共享这个内置的 application 对象。我们已经知道，JSP 引擎为每个客户启动一个线程，也就是说，这些线程共享这个 application 对象。

application 对象的常用方法如下。

(1) public void setAttribute(String key,Object obj)：application 对象可以调用该方法将参数 Object 指定的对象 obj 添加到 application 对象中，并为添加的对象指定了一个索引关键字，如果添加的两个对象的关键字相同，则先前添加的对象被清除。

(2) public Object getAttibute(String key)：获取 application 对象含有的关键字是 key 的对象。由于任何对象都可以添加到 application 对象中，因此用该方法取回对象时，应强制转化为原来的类型。

(3) public Enumeration getAttributeNames()：application 对象调用该方法产生一个枚举对象，该枚举对象使用 nextElemets()遍历 application 对象所含有的全部对象。

(4) public void removeAttribute(String key)：从当前 application 对象中删除关键字是 key 的对象。

(5) public String getServletInfo()：获取 Servlet 编译器的当前版本的信息。

【例 4.7】 利用 application 对象编写一个简单网站计数器。

文件 test_application.jsp 的内容如下：

```
<%@ page contentType="text/html;charset=gb2312"%>
<html><head><title>网页计数器</title></head>
<body>
<%
    if (application.getAttribute("counter")==null)
        application.setAttribute("counter","1");
    else{
      String strnum=null;
      strnum=application.getAttribute("counter").toString();
```

```
        int icount=0;
        icount=Integer.parseInt(strnum);
        icount++;
        application.setAttribute("counter",Integer.toString(icount));
} %>
您是第<%=application.getAttribute("counter")%>位访问者!
</body>
</html>
```

程序运行结果如图 4.13 所示。



图 4.13　简单网站计数器运行结果

4.8　page 对象

page 对象是 java.lang.Object 类的一个实例，通过 page 对象可以访问 JSP 页面本身。

【例 4.8】　page 对象演示实例。

文件 test1.jsp 的内容如下：

```
<%@ page contentType="text/html;charset=GBK" %>
<%
    pageContext.setAttribute("name","zhangsan");
    out.println("test1.jsp: ");
    out.println(pageContext.getAttribute("name"));
    out.println("<p>");
    pageContext.include("test2.jsp");
%>
```

文件 test2.jsp 的内容如下：

```
<%
    out.println("test2.jsp: ");
    out.println(pageContext.getAttribute("name"));
%>
```

运行结果如图 4.14 所示。

说明保存在 pageContext 对象中的属性具有 page 范围，只能在同一个页面中被访问。



图 4.14 page 对象运行结果

4.9 exception 对象页面

exception 对象是 java.lang.Throwable 类的一个实例。它指的是运行时的异常，也就是被调用的错误页面的结果。

文件 application.jsp 的内容如下：

```
<%@ page contentType="text/html; charset=gb2312" language="java"  errorPage="error.
jsp" %>
<html>
<head>
<title>出错演示</title>
</head>
<body>
<%
    int i=1/0;
%>
</body>
</html>
```

文件 error.jsp 的内容如下：

```
<%@ page contentType="text/html; charset=gb2312" language="java" isErrorPage="true"
%>
<html>
<head>
<title>出错了!</title>
</head>
<body>
出错了!<br>
发生了以下的错误:
<br><hr><font color=red>
<%=exception.getMessage()%>
</font></body>
</html>
```

程序运行结果如图 4.15 所示。



图 4.15　exception 对象运行结果

4.10　本章小结

本章主要介绍了常用的 JSP 内置对象，并结合具体实例掌握 JSP 内置对象的用法，为进一步学习 JSP 的强大功能做好了充分准备。

第5章

在 JSP 中使用 JavaBean

本章要点：

- JavaBean 的概念
- JavaBean 的缩写
- JavaBean 的使用

5.1 JavaBean 的概念

在谈论组件之前，让我们看一件通俗的事情：组装电视机。组装一台电视机时，人们可以选择多个组件，例如电阻、电容、显像管等，一个组装电视机的人不必关心显像管是怎么研制的，只要根据说明书了解其中的属性和功能就可以了。不同的电视机可以安装相同的显像管，显像管的功能完全相同，但它们是在不同的电视机里面，一台电视机的显像管发生了故障并不影响其他电视机；也可能两台电视安装了一个共享的组件：天线，如果天线发生了故障，两台电视机都受到同样的影响。

“可视化组件编程”非常成功的一个例子就是微软公司的 VB。人们在使用 VB 编写程序时，经常把一个按钮组件或文本框组件拖放到应用程序窗体中，并了解这个按钮的名字、它有哪些功能、方法等，而且还可以重新更改它的名字。当创建生成应用程序时，这个按钮的名字被保存了下来。但是，微软的组件只适用于微软的操作平台上，不能为其他平台所使用。

按照 Sun 公司的定义，JavaBean 是一个可重复使用的软件组件。实际上，JavaBean 是一种 Java 类，通过封装属性和方法成为具有某种功能或者处理某个业务的对象，简称 beans。由于 JavaBean 是基于 Java 语言的，因此 JavaBean 不依赖平台，具有以下特点。

（1）可以实现代码的重复利用。

（2）易编写、易维护、易使用。

（3）可以在任何安装了 Java 运行环境的平台上使用，而不需要重新编译。

我们已经知道，一个基本的 JSP 页面就是由普通的 HTML 标签和 Java 程序片组成，如果程序片和 HTML 大量交互在一起，页面就显得混杂、不易维护。JSP 页面应当将数据的处理过程指派给一个或几个 beans 来完成，我们只需在 JSP 页面中调用这个 beans 即可。我们不提倡大量的数据处理都用 Java 程序片来完成。在 JSP 页面中调用 beans，可有效地分离静态工作部分和动态工作部分。

5.2　编写 beans

JavaBean 分为可视组件和非可视组件。在 JSP 中主要使用非可视组件。对于非可视组件，我们不必去设计它的外观，主要关心它的属性和方法。

编写 JavaBean 就是编写一个 Java 的类，所以你只要会写类就能编写 beans，这个类创建的一个对象称作一个 beans。为了能让使用这个 beans 的应用程序构建工具(比如 JSP 引擎)知道这个 beans 的属性和方法，只需在类的方法命名上遵守以下规则。

(1) 如果类的成员变量的名字是×××，那么为了更改或获取成员变量的值，即更改或获取属性，在类中就需要有以下两个方法。

① get×××()：用来获取属性×××。

② set×××()：用来修改属性×××。

(2) 对于 boolean 类型的成员变量，即布尔逻辑类型的属性，允许使用“is”代替上面的“get”和“set”。

(3) 类中的普通方法不适合上面的命名规则，但这个方法必须是 public 的。

(4) 类中如果有构造方法，那么这个构造方法也是 public 的，并且是无参数的。

下面我们编写一个简单的 beans，并说明在 JSP 中怎样使用这个 beans。

```
Circle.java:
import java.io.*;
public class Circle {
   int radius;
   public Circle() {
      radius=1;
   }
   public int getRadius() {
      return radius;
   }
   public void setRadius(int newRadius) {
      radius=newRadius;
   }
   public double circleArea() {
      return Math.PI * radius * radius;
   }
   public double circlLength() {
      return 2.0 * Math.PI * radius;
   }
}
```

将上述 Java 文件保存为 Circle.java，并编译通过，得到字节码文件 Circle.class。

5.3　使用 beans

在 JSP 中专门提供三个页面指令来和 JavaBean 交互，分别是 jsp:useBean 指令、jsp:setProperty 指令和 jsp:getProperty 指令。

（1）jsp：useBean 指令的功能是指定 JSP 页面中包括的 JavaBean，具体语法格式如下：

```
<jsp:useBean id="beanid" scope="page|request|session|application" class="package.class"/>
```

其中：

① id：当前页面中引用 JavaBean 的名字，JSP 页面中的 Java 代码将使用这个名字来访问 JavaBean。

② scope：指定 JavaBean 的作用范围，可以取 4 个值。

③ page：JavaBean 只能在当前页面中使用。在 JSP 页面执行完毕后，该 JavaBean 将会被进行垃圾回收。

④ request：JavaBean 在相邻的两个页面中有效。

⑤ session：JavaBean 在整个用户会话过程中都有效。

⑥ application：JavaBean 在当前整个 Web 应用的范围内有效。

（2）jsp:setProperty 指令的功能是设置 JavaBean 的属性。具体语法格式如下：

```
<jsp:setProperty name=" beanName "  propertyDetails>
```

其中，propertyDetails 可以是下面两个值中的一个。

```
property= " *  "
property= "propertyName " param=paramName
```

（3）jsp:getProperty 指令的功能是得到某个 JavaBean 的属性值。

5.4 实践任务：简单的计算器

1. 任务说明

本任务采用 JSP+JavaBean 的模式，实现一个简单计算器的功能，如图 5.1 所示。

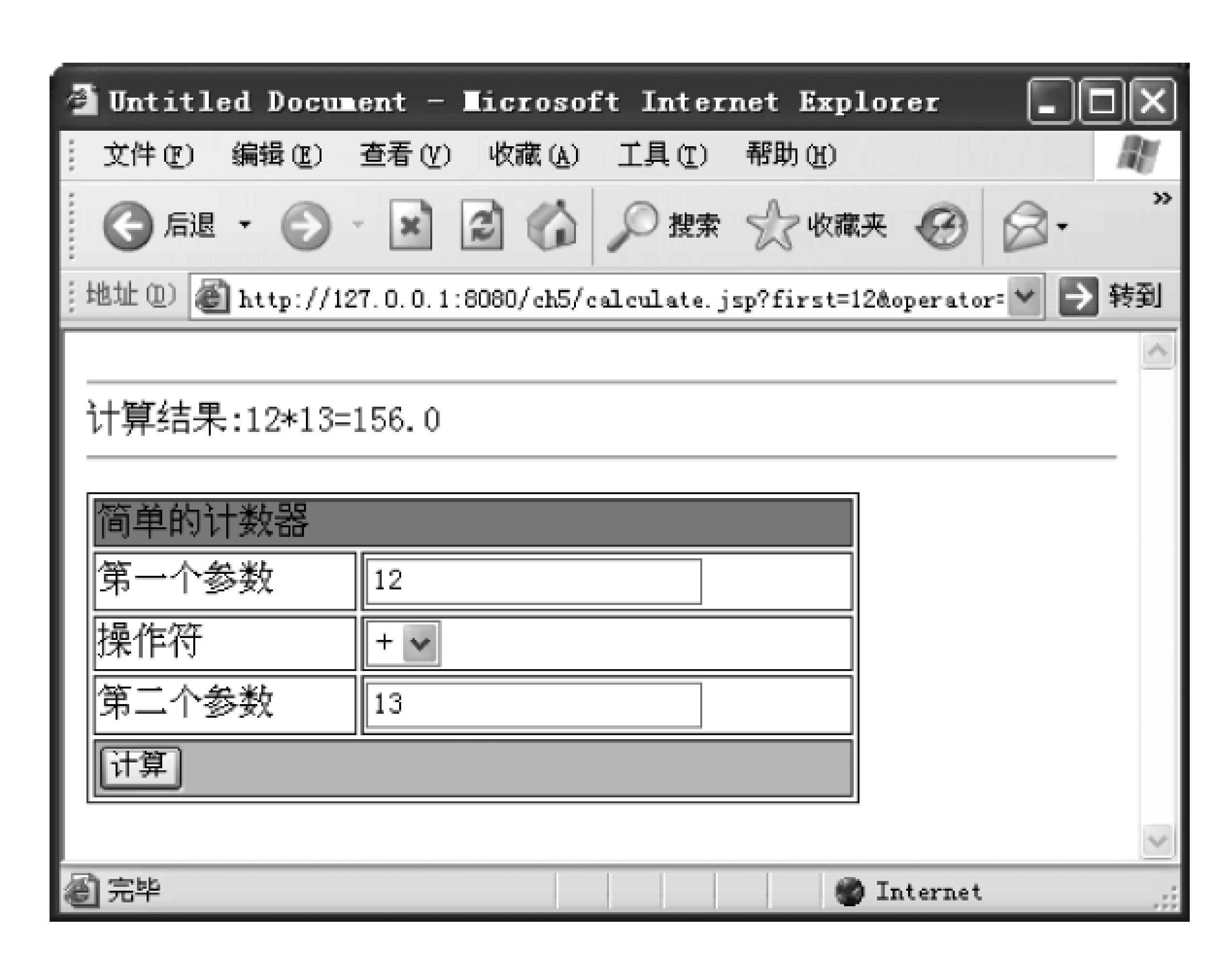


图 5.1 简单的计算器

2. 任务实施

(1) 编写一个 Java 源程序，文件名为 SimpleCalculator.java，内容如下：

```
package com;
public class SimpleCalculator
{
   //属性声明
    private String first;          //第一个操作数
    private String second;         //第二个操作数
    private double result;         //操作结果
    private String operator;       //操作符
    /**
     *以下是一些属性方法
     */
    public void setFirst(String first){
        this.first=first;
    }
    public void setSecond(String second){
        this.second=second;
    }
    public void setOperator(String operator){
        this.operator=operator;
    }
    public String getFirst(){
        return this.first;
    }
    public String getSecond(){
        return this.second;
    }
    public String getOperator(){
        return this.operator;
    }
    //获得计算结果
    public double getResult(){
        return this.result;
    }
    /**
     *根据不同的操作符进行计算
     */
    public void calculate(){
        double one=Double.parseDouble(first);
        double two=Double.parseDouble(second);
        try{
            if(operator.equals("+")) result=one+two;
            else if(operator.equals("-"))result=one-two;
            else if(operator.equals("*"))result=one*two;
            else if(operator.equals("/"))result=one/two;
         }catch(Exception e){
          System.out.println(e);
```

```
            }
        }
}
```

(2) 编译 SimpleCalculator.java 文件，生成 SimpleCalculator.class。

(3) 编写一个 JSP 网页，文件名为 calculate.jsp，内容如下：

```
<%@ page contentType="text/html; charset=gb2312"
language="java"  errorPage="" %>
<jsp:useBean id="calculator"
scope="request" class="com.SimpleCalculator">
<jsp:setProperty name="calculator" property="*"/>
</jsp:useBean>
<html>
<head>
<title>Untitled Document</title>
<meta http-equiv="Content-Type"
content="text/html; charset=gb2312">
</head>
<body>
<hr>
计算结果:<%
try{
    calculator.calculate();
    out.println(calculator.getFirst()+calculator.getOperator()+calculator.getSecond()+
    "="+calculator.getResult());
}
catch(Exception e){
    out.println(e.getMessage());
}
%>
</hr>
<form action="calculate.jsp" method=get>
<table width="75%" border="1" bordercolor="#003300">
  <tr bgcolor="#999999">
    <td colspan="2">简单的计数器</td>
  </tr>
  <tr>
    <td>第一个参数</td>
    <td><input type=text name="first"></td>
  </tr>
  <tr>
    <td>操作符</td>
    <td><select name="operator">
            <option value="+">+</option>
             <option value="-">-</option>
             <option value="*">*</option>
             <option value="/">/</option>
    </select>
    </td>
```

```
  </tr>
  <tr>
    <td>第二个参数</td>
    <td><input type=text name="second"></td>
  </tr>
  <tr>
     <td colspan="2" bgcolor="#CCCCCC"><input type=submit
     value=计算></td>
  </tr>
</table>
</form>
</body>
</html>
```

3. 任务总结

该实践任务采用 JSP+JavaBean 的模式实现了一个简单的计算器应用。通过此种模式，避免了程序片和 HTML 大量交互在一起，而是通过 JavaBean 实现了计算的功能，在 JSP 页面中调用 beans，可有效地分离静态工作部分和动态工作部分。

4. 拓展任务

当客户访问 getNumber.jsp 页面时，随机获取一个 1 到 100 之间的整数，由客户去猜测这个数是多少。客户所猜测的数字如果猜大了或者猜小了，都要给出相应的提示。

5.5　本章小结

本章主要介绍了 JavaBean 组件的使用。程序中往往有重复使用的段落，JavaBean 就是为了能够重复使用而设计的程序段落，而且这些段落并不只服务于某一个程序，而且每个 JavaBean 都具有特定功能，当需要这个功能的时候就可以调用相应的 JavaBean。从这个意义上来讲，JavaBean 大大简化了程序的设计过程，也方便了其他程序的重复使用。JSP +JavaBean 和 JSP +JavaBean + Servlet 成为当前开发 Java Web 应用的主流模式。

第6章

Servlet 技术应用

Web 技术成为当今主流的互联网 Web 应用技术之一，而 Servlet 是 Java Web 技术的核心基础。因而，掌握 Servlet 的工作原理是成为一名合格的 Java Web 技术开发人员的基本要求。本文将带你认识 Java Web 技术是如何基于 Servlet 工作，你将以 Tomcat 为例，了解 Servlet 容器是如何工作的？一个 Web 工程在 Servlet 容器中是如何启动的？Servlet 容器如何解析你在 web.xml 中定义的 Servlet？用户的请求是如何被分配给指定的 Servlet 的？Servlet 容器如何管理 Servlet 生命周期？你还将了解到最新的 Servlet 的 API 的类层次结构，以及 Servlet 中一些难点问题的分析。

本章要点：

- Servlet 简介
- Servlet 技术的特点
- Servlet 的生命周期
- 开发、部署一个简单的 Servlet
- HttpServlet 相关对象的方法列表

6.1　Servlet 简介

Servlet 与 Servlet 容器的关系有点像枪和子弹的关系。枪是为子弹而生，而子弹又让枪有了杀伤力。虽然它们是彼此依存的，但是又相互独立发展，这一切都是为了适应工业化生产的结果。从技术角度来说，是为了解耦，通过标准化接口来相互协作。既然接口是连接 Servlet 与 Servlet 容器的关键，那我们就从它们的接口说起。

Servlet 容器作为一个独立发展的标准化产品，目前它的种类很多，但是它们都有自己的市场定位，很难说谁优谁劣，各有特点。例如，现在比较流行的 Jetty，在定制化和移动领域有不错的发展，我们这里还是以大家最为熟悉的 Tomcat 为例来介绍 Servlet 容器如何管理 Servlet。Tomcat 本身也很复杂，我们只从 Servlet 与 Servlet 容器的接口部分开始介绍。

Tomcat 的容器等级中，Context 容器是直接管理 Servlet 在容器中的包装类 Wrapper，所以 Context 容器如何运行将直接影响 Servlet 的工作方式。

6.1.1　什么是 Servlet

Servlet 是一个 Java 编写的程序，此程序是基于 HTTP 协议的，在服务器端运行的（如 tomcat），是按照 Servlet 规范编写的一个 Java 类。

Servlet 是使用 Java Servlet 应用程序设计接口（API）及相关类和方法的 Java 程序。除了 Java Servlet API，Servlet 还可以使用用以扩展和添加到 API 的 Java 类软件包。Servlet 在启用 Java 的 Web 服务器上或应用服务器上运行，并扩展了该服务器的能力。Java Servlet 对于 Web 服务器，就好像 Java applet 对于 Web 浏览器。Servlet 装入 Web 服务器并在 Web 服务器内执行，而 applet 装入 Web 浏览器并在 Web 浏览器内执行。Java Servlet API 定义了一个 Servlet 和 Java 使能的服务器之间的一个标准接口，这使得 Servlets 具有跨服务器平台的特性。

Servlet 通过创建一个框架来扩展服务器的能力，以提供在 Web 上进行请求和响应服务。当客户机发送请求至服务器时，服务器可以将请求信息发送给 Servlet，并让 Servlet 建立起服务器返回给客户机的响应。当启动 Web 服务器或客户机第一次请求服务时，可以自动装入 Servlet。装入后，Servlet 继续运行，直到其他客户机发出请求。Servlet 的功能涉及范围很广。例如，Servlet 可完成如下功能。

（1）创建并返回一个包含基于客户请求性质的动态内容的完整 HTML 页面。

（2）创建可嵌入现有 HTML 页面中的一部分 HTML 页面（HTML 片段）。

（3）与其他服务器资源（包括数据库和基于 Java 的应用程序）进行通信。

（4）用多个客户机处理连接，接收多个客户机的输入，并将结果广播到多个客户机上。例如，Servlet 可以是多参与者的游戏服务器。

（5）当允许在单连接方式下传送数据的情况下，在浏览器上打开服务器至 applet 的新连接，并将该连接保持在打开状态。当允许客户机和服务器简单、高效地执行会话的情况下，applet 也可以启动客户浏览器和服务器之间的连接。可以通过定制协议或标准（如 IIOP）进行通信。

（6）对特殊的处理采用 MIME 类型过滤数据，例如，图像转换和服务器端包括（SSI）。

（7）将定制的处理提供给所有服务器的标准例行程序。例如，Servlet 可以修改如何认证用户。

6.1.2　Servlet 技术的特点

最早支持 Servlet 技术的是 JavaSoft 的 Java Web Server。此后，一些其他基于 Java 的 Web Server 开始支持标准的 Servlet API。

Servlet 是 Java 技术对 CGI 编程的回答。Servlet 程序在服务器端运行，动态地生成 Web 页面。与传统的 CGI 和许多其他类似 CGI 的技术相比，Java Servlet 具有更高的效率、更容易使用、功能更强大，具有更好的可移植性、更节省投资（更重要的是，Servlet 程序员收入要比 Perl 程序员高）。

1. 高效

在传统的 CGI 中，每个请求都要启动一个新的进程，如果 CGI 程序本身的执行时间较短，启动进程所需要的开销很可能反而超过实际执行时间。而在 Servlet 中，每个请求由一个轻量级的 Java 线程处理(而不是重量级的操作系统进程)。在传统 CGI 中，如果有 *N* 个并发的对同一 CGI 程序的请求，则该 CGI 程序的代码在内存中重复装载了 *N* 次；而对于 Servlet，处理请求的是 *N* 个线程，只需要一份 Servlet 类代码。在性能优化方面，Servlet 也比 CGI 有着更多的选择，比如缓冲以前的计算结果，保持数据库连接的活动等。

2. 方便

Servlet 提供了大量的实用工具例程，例如自动地解析和解码 HTML 表单数据、读取和设置 HTTP 头、处理 Cookie、跟踪会话状态等。

3. 功能强大

在 Servlet 中，许多使用传统 CGI 程序很难完成的任务都可以轻松地完成。例如，Servlet 能够直接和 Web 服务器交互，而普通的 CGI 程序不能。Servlet 还能够在各个程序之间共享数据，使得数据库连接池之类的功能很容易实现。

4. 可移植性好

Servlet 用 Java 编写，Servlet API 具有完善的标准。因此，为 I-Planet Enterprise Server 写的 Servlet，无需任何实质上的改动即可移植到 Apache、Microsoft IIS 或者 WebStar。几乎所有的主流服务器都直接或通过插件支持 Servlet。

5. 节省投资

不仅有许多廉价甚至免费的 Web 服务器可供个人或小规模网站使用，而且对于现有的服务器，如果它不支持 Servlet 的话，要加上这部分功能也往往是免费的(或只需要极少的投资)。

6.1.3 Servlet 的生命周期

Servlet 是一种可以在 Servlet 容器中运行的组件，那么理所当然，就应该有一个从创建到销毁的过程。这个过程我们可以称之为 Servlet 生命周期。Servlet 的生命周期可以分为加载、实例化、初始化、处理客户请求和卸载 5 个阶段，体现在方法上主要是 init()、service()和 destroy()这 3 个方法。生命周期的具体方法说明如下。

Servlet 容器完成加载 Servlet 类和实例化一个 Servlet 对象，init()方法完成初始化工作，该方法由 Servlet 容器调用完成，service()方法处理客户端请求，并返回响应结果，destroy()方法在 Servlet 容器卸载 Servlet 之前被调用，释放一些资源。

(1) init()方法：服务器加载和初始化 Servlet。

Servlet 引擎负责加载和初始化 Servlet，这个过程可以在 Servlet 引擎加载时执行，可以在 Servlet 响应请求时执行，也可以在两者之间的任何时候执行。

Servlet 引擎加载好 Servlet 后，必须初始化它。初始化时，Servlet 可以从数据库里读取初始数据，建立 JDBC 连接，或者建立对其他有价值的资源的引用。

在初始化阶段，init()方法被调用。这个方法在 javax. servlet. Servlet，接口中定义。init()方法以一个 Servlet 配置文件为参数。Servlet Config 对象由 Servlet 引擎实现，可以让 Servlet 从中读取一些 name-value 对应的参数值。Servlet Config 对象还可以让 Servlet 接受一个 Servlet Context 对象。

（2）service()方法：初始化完毕，Servlet 对象调用该方法获得相应客户的请求。

Servlet 被初始化以后，就处于能响应请求的就绪状态。每个对 Servlet 的请求由一个 ServletRequest 对象代表。Servlet 给客户端的响应由一个 ServletResponse 对象代表。当客户端有一个请求时，Servlet 引擎将 ServletRequest 和 ServletResponse 对象都转发给 Servlet，这两个对象以参数的形式传给 service 方法。这个方法由 javax. servlet. Servlet 定义，并由具体的 Servlet 实现。

Servlet 还可以实现 ServletRequest 和 ServletResponse 接口。ServletRequest 接口可以让 Servlet 获取客户端请求中的参数，如表单数据、请求信息、协议类型，等等。Servlet 可以从 ServletInputStream 流中读取请求数据。ServletResponse 接口允许 Servlet 设置响应头信息和状态代码。实现这个接口，可以使 Servlet 能访问 ServletOutputStream 流，用来向客户端返回数据。

（3）destroy()方法：调用该方法销毁 Servlet 对象。

Servlet 引擎没有必要在 Servlet 生命周期的每一段时间内都保持 Servlet 的状态。Servlet 引擎可以随时随意使用或释放 Servlet。因此，不能依赖 Servlet class 或其成员存储信息。当 Servlet 引擎判断一个 Servlet 应当被释放时(比如说，引擎准备关闭或需要回收资源)，引擎必须让 Servlet 能释放其正在使用的任何资源，并保存持续性的状态信息。这些可以通过调用 Servlet 的 destroy 方法实现。在 Servlet 引擎释放一个 Servlet 以前，必须让其完成当前实例的 service 方法，或是等到 timeout(如果引擎定义了 timeout)。当引擎释放一个 Servlet 以后，引擎将不能再将请求转发给它，引擎必须彻底释放该 Servlet，并将其标明为可回收的。

其中，init()方法只在 Servlet 第一次被请求加载的时候被调用一次，当有客户再请求 Servlet 服务时，Web 服务器将启动一个新的线程，在该线程中，调用 service 方法响应客户的请求。

6.1.4　开发、部署一个简单的 Servlet

在 Java Web 应用开发中，创建 HttpServlet 一般遵循以下几个步骤。

（1）自定义一个类，该类继承 HttpServlet。

（2）重写 HttpServlet 类中的 doGet()和 doPost()方法。

（3）获取 Http 请求消息(一般由表单提供的数据)进行业务逻辑处理。

（4）生成 Http 响应结果，以流的形式输出到客户端浏览器或输出到磁盘文件。

（5）在 web. xml 配置文件中注册自定义的 HttpServlet 类。

(6) 映射自定义的 HttpServlet 类的访问。

(7) 启动 Tomcat 容器进行测试。

下面举一个简单的 Servlet 实例,实现 Servlet 的开发和部署。

(1) 自定义类 HelloServlet 并继承 HttpServlet,重写 HttpServlet 类中的 doGet()方法,以实现具体的处理功能,该类具体实现的代码如下:

```
HelloServlet.java
import java.io.*;
import javax.servlet.http.*;
public class HelloServlet extends HttpServlet {
  public void service(HttpServletRequest request, HttpServletResponse response) throws IOException {
    response.setContentType("text/html;charset=gb2312");
    PrintWriter out = response.getWriter();
    out.println("<HTML>");
    out.println("<HEAD>");
    out.println("<TITLE>Hello Servlet</TITLE>");
    out.println("</HEAD>");
    out.println("<BODY>");
    out.println("<B>Hello, World !</B>");
    out.println("</BODY>");
    out.println("</HTML>");
    out.close();
  }
}
```

(2) 在 web.xml 文件中进行注册和映射自定义的 Servlet 类,具体的配置文件信息如下:

```
<servlet>
<servlet-name>HelloServlet</servlet-name>
<servlet-class>HelloServlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>HelloServlet</servlet-name>
<url-pattern> /HelloServlet</url-pattern>
</servlet-mapping>
```

(3) 启动 Tomcat 容器进行测试,在浏览器的地址栏上输入 http://localhost:8080/ServletExample/HelloServlet 以后,运行结果如图 6.1 所示。

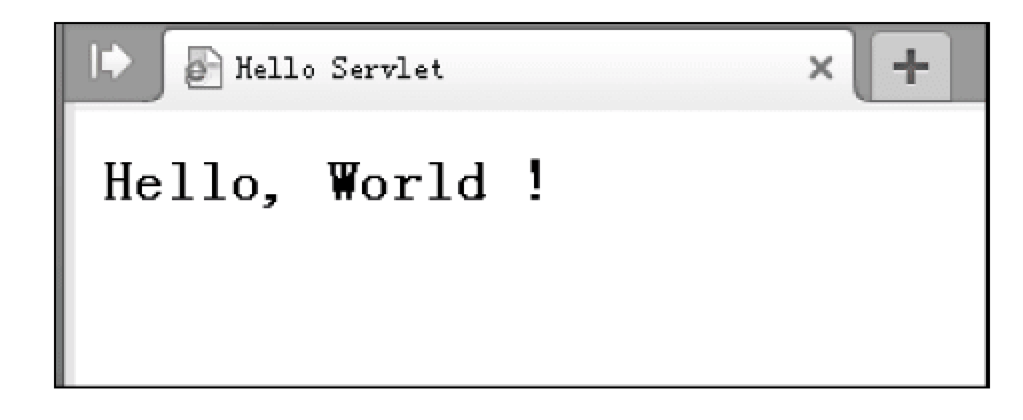


图 6.1 Hello Servelt 运行结果

6.2　HttpServlet 相关对象的方法列表

HttpServlet 是 Servlet 的抽象类，HttpServlet 通过 HttpServletRequest 对象和 HttpServletResponse 对象与客户进行交互。其中 HttpServletRequest 对象负责封装存放 Http 请求消息，HttpServletResponse 对象负责封装存放处理得到的响应结果。

另外 Servlet 容器启动加载 Web 应用时，会为每个 Web 应用创建一个 ServletContext 对象，用于存放整个 Web 应用的共享数据，因此，我们可以把 ServletContext 对象看成 Web 应用的共享内存区。

1. HttpServletRequest 接口

HttpServletRequest 接口中最常用的方法就是获得请求中的信息。在 JSP 中，内置对象 request 是一个 HttpServletRequest 的实例。主要有以下方法。

(1) getCookies()：获得客户端发送的 Cookie。返回一个数组，该数组包含这个请求中当前的所有 Cookie。如果这个请求中没有 Cookie，返回一个空数组。

(2) getSession()：返回和客户端关联的 Session。如果没有，则为空。

(3) getSession(boolean create)：和上一个方法类似，不过，如果没有给客户端分配 Session，则会创建一个新的 Session 并返回。

(4) getParameter(String name)：获得请求中指定参数的值，如果没有，则返回空。

(5) getParameterValues(String name)：返回请求中名为 name 的参数值，这个值往往是 checkbox 或者 select 控件提交的信息，获得的值是一个 String 数组。

2. HttpServletResponse 接口

HttpServletResponse 接口代表对客户端的 HTTP 响应，主要有以下方法。

(1) addCookie(Cookie cookie)：在响应中添加一个 Cookie。

(2) encodeURL(String url)：使用 URL 和一个 SessionId 重写这个 URL。

(3) sendRedirect(String lication)：把响应送到另一个页面或 Servlet 进行处理。

(4) setContentType(String type)：设置响应的 MIME 类型。

(5) setCharacterEncoding(String charset)：设置响应的字符编码。

3. ServletContext 接口

在服务器上使用 session 对象维持与单个客户相关的状态，而当为多个用户的 WEB 应用维持一个状态时，则使用 Servlet 环境(ServletContext)。以下是常用方法。

(1) getAttribute(String name)：获得 ServletContext 中名称为 name 的属性。

(2) getContext(String uripath)：返回给定的 URIPATH 的应用的 Servlet 上下文。例如：

```
ServletContext test = getContext("/test");
```

(3) removeAttribute(String name)：删除名字为 name 的属性。

(4) setAttribute(String name,Object obj):在 ServletContext 中设置一个属性。

6.3 创建 HttpServlet 实例

接下来举一个访问计数器的实例,以此例来加深对 HttpServlet 的理解。

【例 6.1】 访问计数器用户通过填写 index.html 中的表单,提交后,Servlet 窗口将整个 HTTP 请求信息封装到一个 HttpServletRequest 对象中,传入 MyServlet.java 中,由 MyServlet 维护 ServletContext 对象中的一个计数器 count 属性,最后将处理的响应结果封装到一个 HttpServletResponse 对象中,通过 HttpServletResponse 对象的 PrintWriter 输出流对象将响应结果输出到用户的浏览器上。

index.html 文件的代码如下:

```
<html>
  <head>
    <title>测试 Servlet 页面</title>
    <meta http-equiv="content-type" content="text/html; charset=gb2312">
  </head>

  <body>
    <FORM name="form1" action="MyServlet" method="POST">
        请输入姓名:
        <INPUT name="name" type="text">
        <INPUT name="submit" value="提交" type="submit">
    </FORM>
  </body>
</html>
```

MyServlet.java 文件的代码如下:

```
public class MyServlet extends HttpServlet {

    public MyServlet() {                    //构造函数
        super();
    }

    public void destroy() {                 //销毁时将调用的方法
        super.destroy();                    //Just puts "destroy" string in log
        //Put your code here
    }

    //处理以 GET 方式提交过来的表单数据
    public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        doPost(request,response);                       //直接调用 doPost 方法
    }

    //处理以 Post 方式提交上来的表单数据
```

```
    public void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        response.setContentType("text/html");
        response.setCharacterEncoding("gb2312");      //设置响应采用的编码方式
        String name = request.getParameter("name");
        //从提交上来的表单中取 name 文本框的值
        name = new String(name.getBytes("ISO8859-1"),"gb2312");
        //将字符编码转换为 gb2312
        PrintWriter out = response.getWriter();
        ServletContext context = getServletContext();          //得到整个 Web 应用的
                                                                //ServletContext 对象
        int count = 1;                                          //从 1 开始起计
        if (context.getAttribute("count")==null){              //如果是第一位访客
            context.setAttribute("count",new Integer(count));   //设置计数器初始值
        }else{                                                  //如果不是第一位访客
            count = Integer.parseInt(context.getAttribute("count").toString());
                                                                //取出现有的计数值
            count++;                                            //计数加 1
            context.setAttribute("count",new Integer(count));   //更新计数器内容
        }
        out.println("<HTML>");
        out.println("<HEAD><TITLE>A Servlet</TITLE></HEAD>");
        out.println("<BODY>");
        out.println(name+",你好!你是第"+count+"位访客!");  //实现简单问好
        out.println("</BODY>");
        out.println("</HTML>");
        out.flush();
        out.close();
    }

    //本 Servlet 装载到容器后将自动执行的初始化方法
    public void init() throws ServletException {
        //Put your code here
    }
}
```

HttpServlet 开发完成后，必须在 web.xml 文件中配置以后才能被用户通过浏览器访问，MyServlet 在 web.xml 文件中配置信息如下。

web.xml 文件的代码如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4"
xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
  <servlet>
    <servlet-name>MyServlet</servlet-name>
    <servlet-class>MyServlet</servlet-class>
```

```
    </servlet>
    <servlet-mapping>
        <servlet-name>MyServlet</servlet-name>
        <url-pattern>/MyServlet</url-pattern>
    </servlet-mapping>
</web-app>
```

程序编写完毕以后，在浏览器的地址栏上输入 http://localhost:8080/TestServlet/index.html，运行的结果如图 6.2 所示。

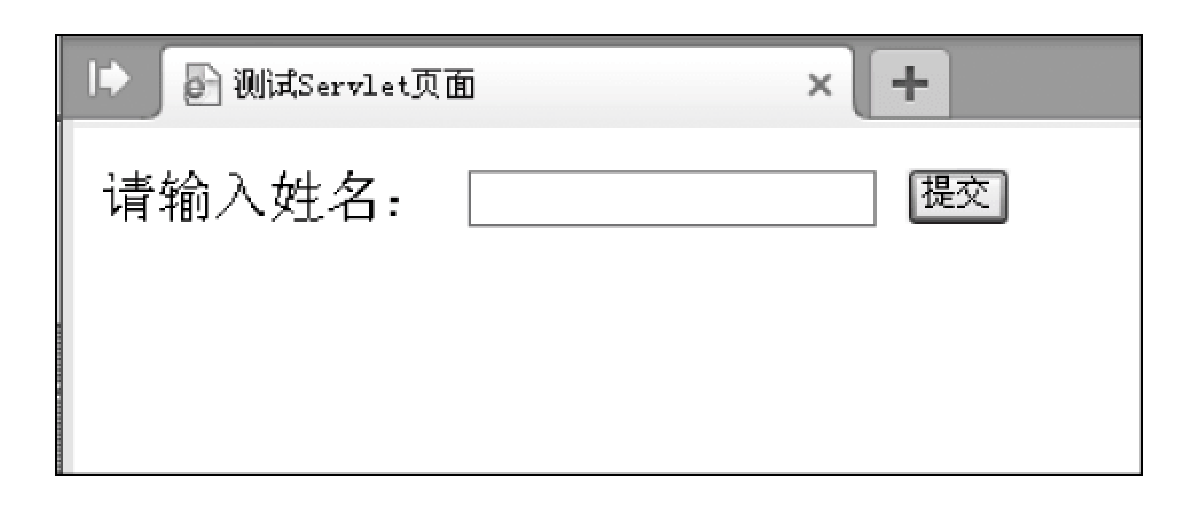


图 6.2 index.html 页面的运行结果

单击图 6.2 中的“提交”按钮，运行的效果如图 6.3 所示。

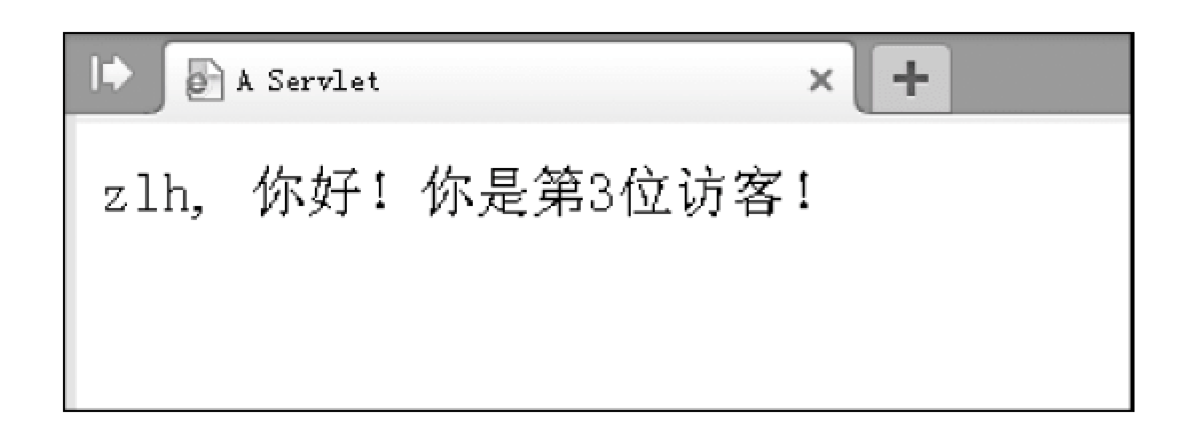


图 6.3 单击“提交”按钮后的运行结果

6.4 拓展任务：学生管理实例

下面通过一个简单的学生管理实例来说明 Servlet 的应用和在 Servlet 中涉及的相关技术。实现一个简单的学生管理的网站，在该系统中有两个表，一个为 usertable 表，另一个为 student 表。

6.4.1 创建数据库

usertable 是用户表，用来存放用户名和密码，表的结构如表 6.1 所示。

表 6.1 用户表结构

字段名称	数据类型	备注
id	int	序号
username	varchar2(10)	用户名
pwd	varchar2(10)	密码

student 是学生表，用来存放学生的基本表，表的结构如表 6.2 所示。

表 6.2　学生表结构

字段名称	数据类型	备　注
id	varchar2(10)	学号
name	varchar2(10)	姓名
age	number	年龄
major	varchar2(20)	专业

6.4.2　设计界面

1. 登录界面

登录界面 login.jsp，如图 6.4 所示。

要求：输入 admin 用户名和 admin 密码后，单击“提交”按钮，进入 selectall.jsp 页面，如果输入不正确的用户名和密码，则跳回 login.jsp 页面。

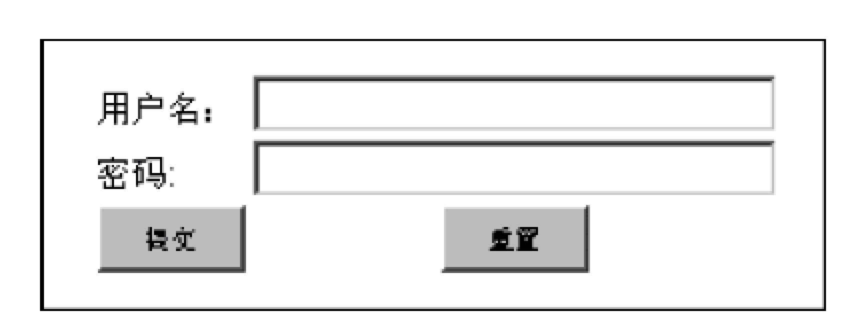


图 6.4　登录界面

2. selectall.jsp 文件的界面

学生信息列表界面 selectall.jsp，如图 6.5 所示。

学号	姓名	年龄	专业	操作
961405	tom	12	食品	删除|修改
961402	李四	21	汽车	删除|修改
961403	王五	22	计算机应用	删除|修改
961409	姜华	24	计算机应用	删除|修改
961402	李四	21	汽车	删除|修改
961403	王五	22	计算机应用	删除|修改

添加

图 6.5　selectall.jsp 文件的界面

要求：

(1) 运行 selectall.jsp 文件将把学生表中的所有数据从数据库中提取出并按图 6.5 进行显示。

(2) 单击“删除”超链接将文件中以学号为关键字的记录删除。

(3) 单击“修改”超链接进入 updatestudent.jsp 文件，并把该条记录的内容显示在此页面中。

(4) 单击“添加”超链接进入 addstudent.jsp 文件中。

3. addstudent.jsp 文件的界面

添加学生信息界面 addstudent.jsp，如图 6.6 所示。

要求：单击“提交”按钮把填写的记录添加到数据库中。之后跳转到 selectall.jsp 文件中。

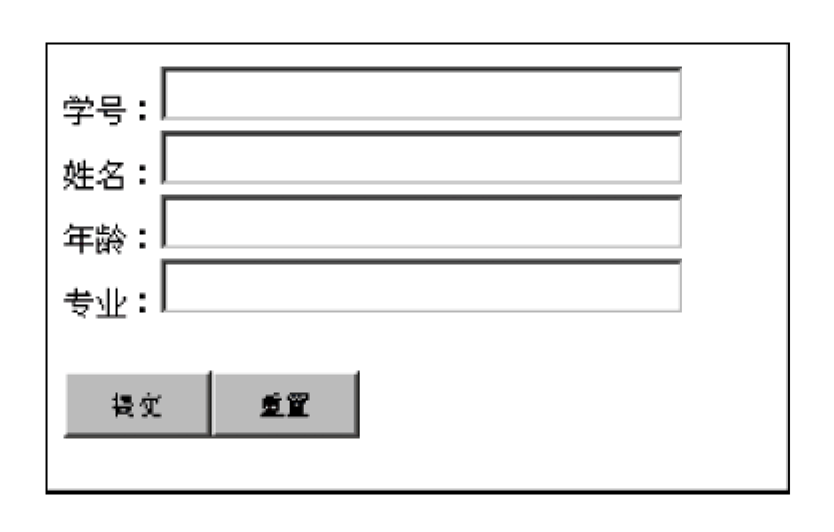




图 6.6　addstudent.jsp 文件的界面

图 6.7　updatestudent.jsp 文件的界面

4. updatestudent.jsp 文件的界面

更新学生信息界面 updatestudent.jsp，如图 6.7 所示。

要求：

(1) 接收 selectall.jsp 文件传过的信息，并显示在此文件中。

(3) 学号为只读。

(3) 单击“提交”按钮把修好后的数据写到数据库中。之后跳转到 selectall.jsp 文件中。

6.4.3　类设计

1. Student 类

Student 类用于封装学生的基本信息，存放在 cvit.com.bean 包下。

成员变量如下：

```
String id;
String name;
int age;
String major;
```

成员方法如下：

```
public String getId();
public String getName();
public int getAge();
public String getMajor();
public void setId(String id);
public void setName(String name);
public void setAge(int age);
public void setMajor(String major);
```

2. Dao 类

Dao 类提供链接数据库、发送 SQL 语句和关闭数据库链接的功能，存放在 cvit.com.dao 包中。

成员变量如下：

```
Connection connl;
Statement stmt;
ResultSet rsl;
```

成员方法如下：

```
public Connection getConn();              //负责连接数据库
public void execute(String sql);          //负责执行 SQL 语句
public void close();                      //负责关闭资源
```

3. StudentManager 类

StudentManager 类提供对学生基本信息的查询、修改、删除和添加功能，存放在 cvit.com.dao 包中。

成员方法如下：

```
public ArrayList findAll();               //查出所有学生信息存放在 ArrayList 集合中
public void delete(String id);            //删除以 id 为学号的学生
public void add(Student s);               //添加一个学生信息
public Student queryOne(String id)        //查询学号为 id 的一个学生，并返回该 Student 对象
```

4. UserName 类

UserName 类提供对用户的查询功能，存放在 cvit.com.dao 包中。其原型如下：

```
public boolean isUser(String username,String pwd);
//根据用户和密码判断是不是合法用户.是合法用户,返回 true; 不是合法用户,返回值为 false.
```

6.4.4　编写 Servlet 类

(1) AddStudentServlet 类：负责接收 addstudent.jsp 文件传递的参数，把信息封装成一个 Student 对象，StudentManager 对象调用相应的方法把信息添加到数据库中，之后负责跳转到 selectall.jsp 文件中。

(2) DeleteStudentServlet 类：负责接收 selectall.jsp 传递的 id 学号参数，StudentManager 对象调用相应的方法把以 id 为关键字的学号删除，之后负责跳转到 selectall.jsp 文件中。

(3) UpdateStudentServlet 类：负责接收 updatestudent.jsp 文件中传递的参数，把信息封装成一个 Student 对象，StudentManager 对象调用相应的方法把信息添加到数据库中，之后负责跳转到 selectall.jsp 文件中。

(4) UserServlet 类：负责接收 login.jsp 文件传递的参数，UserName 对象调用相应的方法判断是不是合法用户，是合法用户跳到 selectall.jsp 文件中，不是合法用户跳到 login.jsp 文件中。

6.4.5　JSP 文件代码

1. login.jsp 文件

具体代码如下：

```
<%@page contentType="text/html; charset=GBK"%>
<html>
<head>
<title>登录</title>
</head>
<body bgcolor="#ffffff">
<table align="center">
  <form method="post" action="userservlet">
  <!--表单把信息提交到 userservlet 对应的 servlet 中-->
  <tr>
    <td>用户名: </td>
    <td>
      <input type="text" name="username" value=""/>
    </td>
  </tr>
  <tr>
    <td>密码:</td>
    <td>
      <input type="password" name="pwd" value=""/>
    </td>
  </tr>
  <tr>
    <td>
      <input type="submit" name="Submit" value="提交">
    </td>
    <td align="center">
      <input type="reset" value="重置">
    </td>
  </tr>
  </form>
</table>
</body>
</html>
```

2. selectall.jsp 文件

具体代码如下：

```
<%@ page contentType="text/html; charset=GBK" %>
<%@ page import="java.util.ArrayList*,cvit.com.bean.Student"%>
<%@ page import="cvit.com.dao.StudentManager"%>
<!--该处引入 ArrayList、Student 和 StudentManager 类-->
<html>
<head>
<title>
显示所有学生信息
</title>
</head>
<body bgcolor="#ffffff">
<table width="80%" border="1">
```

```
  <tr align="center">
    <td>学号</td>
    <td>姓名</td>
    <td>年龄</td>
    <td>专业</td>
    <td>操作</td>
  </tr>
  <%
    //创建一个 StudentManager 对象.
    StudentManager sm=new StudentManager();
    //查出所有学生的信息存放到 ArrayList 对象 a 中
    ArrayList a=sm.findAll();
    //遍历 a 对象获得每个 student 的信息
    for(int i=0;i<a.size();i++){
      //获得 a 对象当前存放的对象,并转换为 Student 类型的对象
      Student s=(Student)a.get(i);
  %>
  <tr align="center">
    <td><%=s.getId() %></td>
    <td><%=s.getName() %></td>
    <td><%=s.getAge()%></td>
    <td><%=s.getMajor() %></td>
    <td>
    <a href="deletestudentservlet?id=<%=s.getId()%>">删除</a>|
    <a href="updatestudent.jsp?id=<%=s.getId()%>">修改</a>
    <!--此处设置删除超链接分别指向 deletestudentservlet 对象的 servlet,修改的超链接指向
    updatestudent.jsp 文件.添加的超链接指向 addstudent.jsp 文件-->
    </td>
  </tr>
  <%
    }
  %>
</table>
<a href="addstudent.jsp">添加</a>
</body>
</html>
```

3. addstudent.jsp 文件

具体代码如下：

```
<%@ page contentType="text/html; charset=GBK" %>
<html>
<head>
</head>
<body bgcolor="#ffffff">
<!--表单把信息提交到 addstudentservlet 对应的 servlet 中-->
 <form method="post" action="addstudentservlet">
   学号: <input type="text" name="id" value="" /><br>
   姓名: <input type="text" name="name" value=""/><br>
```

```
    年龄：<input type="text" name="age" value=""/><br>
    专业：<input type="text" name="major" value=""/><br>
<br>
<input type="submit" name="Submit" value="提交">
<input type="reset" value="重置">
</form>
</body>
</html>
<%@ page contentType="text/html; charset=GBK" %>
<%@ page import="java.util.*,cvit.com.bean.Student" %>
<%@ page import="cvit.com.dao.StudentManager" %>
<!--该处引入 ArrayList、Student 和 StudentManager 类-->
<html>
<head>
</head>
<body bgcolor="#ffffff">
 <%
    //获得超链接传递的参数
    String id=request.getParameter("id");
    //创建一个 StudentManager 对象
    StudentManager sm=new StudentManager();
    //查找学号为 id 的学生,存放到 Student 对象 s 中
    Student s=sm.queryOne(id);
 %>
 <!--表单把信息提交到 updatestudentservlet 对应的 servlet 中-->
 <form method="post" action="updatestudentservlet">
    学号：<input type="text" name="id" value="<%=s.getId()%>" readonly/><br>
    姓名：<input type="text" name="name" value="<%=s.getName()%>"/><br>
    年龄：<input type="text" name="age" value="<%=s.getAge()%>"/><br>
    专业：<input type="text" name="major" value="<%=s.getMajor()%>"/><br>
 <br>
 <input type="submit" name="Submit" value="提交">
 <input type="reset" value="重置">
</form>
</body>
</html>
```

以登录这件事务来说明 JSP、Servlet、JavaBean 之间的关系。首先，在 login.jsp 文件中输入数据，提交时，会查表单中 action 属性对应的值，查到的值为 userservlet，就到 web.xml 找到 userservlet 对应的 cvit.com.servlet.UserServlet 类，服务器启动一个 UserServlet 实例；在 UserServlet 实例中通过 request 对象的 getParameter()方法获得表单中参数 username 和 pwd 值，之后服务器创建一个 UserName 实例，UserName 实例中调用 isUser()方法，isUser()方法将访问数据库，判断 username 和 pwd 是不是 usertable 表中存储的记录，如果是返回 true，否则返回 false。返回 true，response 对象调用 sendRedirect()方法，跳到 selectall.jsp 文件中；返回 false，response 对象调用 sendRedirect()方法，跳到 login.jsp 文件中。在这个过程中，login.jsp 文件负责前台页面显示，接受数据和发送数据，UserName 类负责访问数据库，UserServlet 类负责接收

login.jsp 文件传递的参数，调用 UserName 类，最后再把结果给 selectall.jsp 页面显示，Servlet 在这个过程中将起到控制调动的功能。

6.5 本章小结

在这一章中，我们主要介绍了有关 Java Servlet 的基础知识、Servlet 的实际应用以及 Servlet 的基本结构。到此，对于 Servlet 请求的流向情况、什么对象被调用有了更深入的理解。还介绍了 Servlet 的生命周期，也剖析了一个基本的 Servlet，使我们对一个 Servlet 的组成及概况有了一定的了解，也知道了自己的 Servlet 的哪一部分需要符合 Java Servlet 的框架结构要求，怎样组合 Servlet 和 JSP，以实现服务器端的 Web 应用解决方案。

第7章

过　滤　器

Java Web 应用的中文编码问题几乎是所有初学者的棘手难题，在 Servlet 2.3 以前，解决此类问题的方法就是通过硬编码来进行编码转换。Servlet 2.3 新增的过滤器(Filter)功能使编码的转换变得如此简单。

本章要点：

- 什么是过滤器
- 过滤器的工作原理
- 过滤器的配置说明
- 过滤器的实例

7.1　Filter 简介

Filter 也称为过滤器，它是 Servlet 2.3 以上版本新增加的一个功能，其技术也是非常强大的。通过 Filter 技术可以对 Web 服务器的文件进行拦截，从而实现一些特殊的功能。在 JSP 开发应用中也是必备的技能之一。

过滤器是一个程序，它先于与之相关的 Servlet 或 JSP 页面运行在服务器上。过滤器可附加到一个或多个 Servlet 或 JSP 页面上，并且可以检查进入这些资源的请求信息。在这之后，过滤器可以作如下的选择。

(1) 以常规的方式调用资源(即调用 Servlet 或 JSP 页面)。

(2) 利用修改过的请求信息调用资源。

(3) 调用资源，但在发送响应到客户机前对其进行修改。

(4) 阻止该资源调用，代之以转到其他的资源，返回一个特定的状态代码或生成替换输出。

Filter 的基本功能是对 Servlet 容器调用 Servlet 的过程进行拦截，从而在 Servlet 进行响应处理的前后实现一些特殊的功能。在 Servlet API 中定义了三个接口类来供开发人员编写 Filter 程序：Filter、FilterChain、FilterConfig。

Filter 程序是一个实现了 Filter 接口的 Java 类，与 Servlet 程序相似，它由 Servlet 容器进行调用和执行。Filter 程序需要在 web.xml 文件中进行注册和设置它所能拦截的资源。Filter 程序可以拦截 JSP、Servlet、静态图片文件和静态 HTML 文件等。

所有的 Servlet 过滤器都必须实现 javax.servlet.filter 接口，该接口中定义了 3 个过滤器必须实现的方法。

(1) void init(FilterConfig)：过滤器的初始化方法，Servlet 容器在创建过滤器实例时调用这个方法。在这个方法中，可以读出在 web.xml 文件中为该过滤器配置的初始化参数。

(2) void doFilter(ServletRequest，ServletResponse，FilterChain)：用于完成实际的过滤操作。当客户请求访问与过滤器相关联的 URL 时，Servlet 容器将先调用过滤器的这个方法，FilterChain 参数用于访问后续过滤器。

(3) void destroy()：过滤器在被取消前执行这个方法，释放过滤器申请的资源。

Filter 的基本工作原理如下。

当在 web.xml 中注册了一个 Filter 来对某个 Servlet 程序进行拦截处理时，这个 Filter 就成了 Servlet 容器与该 Servlet 程序的通信线路上的一道关卡。该 Filter 可以对 Servlet 容器发送给 Servlet 程序的请求和 Servlet 程序回送给 Servlet 容器的响应进行拦截，可以决定是否将请求继续传递给 Servlet 程序，以及对请求和响应信息是否进行修改。在一个 Web 应用程序中可以注册多个 Filter 程序，每个 Filter 程序都可以对一个或一组 Servlet 程序进行拦截。若有多个 Filter 程序对某个 Servlet 程序的访问过程进行拦截，当针对该 Servlet 的访问请求到达时，Web 容器将把这多个 Filter 程序组合成一个 Filter 链(过滤器链)。Filter 链中各个 Filter 的拦截顺序，与它们在应用程序的 web.xml 中映射的顺序一致。Filter 的工作原理如图 7.1 所示。

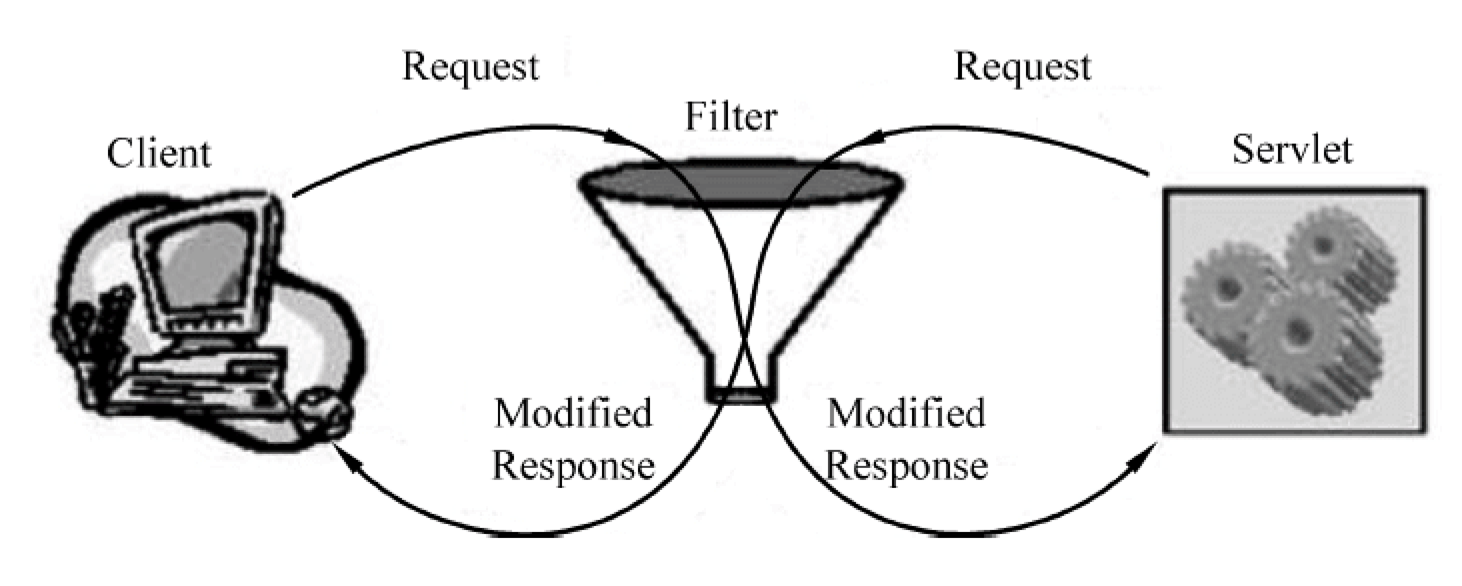


图 7.1　Filter 的工作原理

7.2　Filter 配置说明

在实现一个过滤器类以后，需要在 web.xml 部署文件中对过滤器进行配置，通过 filter 和 filter-mapping 两个元素来完成。

filter 元素用于在 Web 应用程序中声明一个过滤器。在 filter 元素内，filter-name 用于为过滤器指定一个名字，该元素的内容不能为空。filter-class 元素用于指定过滤器的完整限定类名。init-param 元素用于为过滤器指定初始化参数，它的子元素 param-name 指定参数的名字，param-value 指定参数的值。在过滤器中，可以使用 FilterConfig 接口对象来访问初始化参数。

filter-mapping 元素用于指定过滤器关联的 URL 样式或者 Servlet。其中 filter-name 子元素的值必须是在 filter 元素中声明过的过滤器的名字。url-pattern 元素和 servlet-name 元素可以选择一个；url-pattern 元素指定过滤器关联的 URL 样式；servlet-name

元素指定过滤器对应的 Servlet。用户在访问 url-pattern 元素指定的 URL 上的资源或 servlet-name 元素指定的 Servlet 时,该过滤器才会被容器调用。filter-mapping 元素还可以包含 0 到 4 个 dispatcher,指定过滤器对应的请求方式,可以是 REQUEST、INCLUDE、FORWARD 和 ERROR 之一,默认是 REQUEST。

(1) REQUEST:当用户直接访问页面时,Web 容器将会调用过滤器。如果目标资源是通过 RequestDispatcher 的 include()或 forward()方法访问时,那么该过滤器就不会被调用。

(2) INCLUDE:如果目标资源是通过 RequestDispatcher 的 include()方法访问时,那么该过滤器将被调用。除此之外,该过滤器不会被调用。

(3) FORWARD:如果目标资源是通过 RequestDispatcher 的 forward()方法访问时,那么该过滤器将被调用,除此之外,该过滤器不会被调用。

(4) ERROR:如果目标资源是通过声明式异常处理机制调用时,那么该过滤器将被调用。除此之外,过滤器不会被调用。

7.3 使用 Filter 实现编码过滤器

下面以编码过滤器为例,综合讲解一个过滤器在 Java Web 程序中的具体应用。

【例 7.1】 过滤器的应用。首先编制一个编码过滤器类 EncodingFilter.java,然后将编码过滤器 EncodingFilter 在 web.xml 文件中进行配置,再创建一个发起请求的 HTML 测试页面 index.html,最后再创建一个用于处理请求的 JSP 页面 process.jsp。

EncodingFilter.java 文件的内容如下:

```
import java.io.IOException;
import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
public class EncodingFilter implements Filter {
    String encoding = null;
    FilterConfig filterConfig = null;
    //初始化方法
    public void init(FilterConfig filterConfig) throws ServletException {
        //TODO Auto-generated method stub
        this.filterConfig = filterConfig;
        this.encoding = filterConfig.getInitParameter("encoding");
    }
    //过滤处理方法
    public void doFilter(ServletRequest request, ServletResponse response,
    FilterChain chain) throws IOException, ServletException {
        //TODO Auto-generated method stub
```

```
            if (encoding!=null){
                //对请求进行编码设置
                request.setCharacterEncoding(encoding);
            }
            //将处理权转交给下一个处理器
            chain.doFilter(request,response);
        }
        //销毁方法
        public void destroy() {
            //TODO Auto-generated method stub
            this.encoding = null;
            this.filterConfig = null;
        }
    }
```

process.jsp 文件的内容如下：

```
<%@ page language="java" import="java.util.*" pageEncoding="gb2312"%>
<html>
  <head>
    <title>编码过滤器测试</title>
  </head>

  <body>
    <%=request.getParameter("name")%>,你好!<br>
  </body>
</html>
```

web.xml 文件的主要内容如下：

```
<filter>
    <filter-name>encodingFilter</filter-name>
    <filter-class>EncodingFilter</filter-class>
    <init-param>
        <param-name>encoding</param-name>
        <param-value>gb2312</param-value>
    </init-param>
</filter>
<filter-mapping>
    <filter-name>encodingFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

通过编码过滤器以后，处理请求的 JSP 页面可直接获取正常的中文参数值，而不必进行编码的硬性转换，彻底解决了 Java Web 应用的中文编码问题。

本实例的运行结果如图 7.2 和图 7.3 所示。



图 7.2　处理请求页面

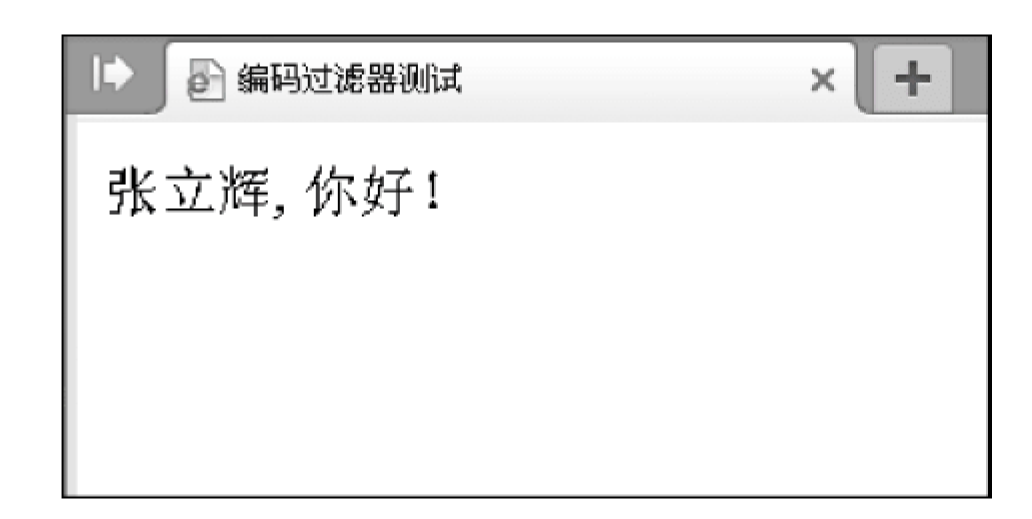


图 7.3　处理结果页面

7.4　使用 Filter 实现计时过滤器

【例 7.2】 计时过滤器实例。为了加深对 Filter 应用的印象，在上个编码过滤器实例的基础上，再增加计时过滤器功能，计时过滤器用于计算从接收请求到生成响应结果所耗费的时间。

PageTimerFilter.java 文件的内容如下：

```
import java.io.IOException;
import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import java.util.*;
public class PageTimersFilter implements Filter {
    FilterConfig filterConfig = null;
    public void init(FilterConfig filterConfig) throws ServletException {
        this.filterConfig = filterConfig;
    }
    public void doFilter(ServletRequest request, ServletResponse response,
            FilterChain chain) throws IOException, ServletException {
        Date startTime, endTime;
        double duration = 0;
        startTime = new Date();                          //记下开始时间
        chain.doFilter(request, response);               //将处理权转交给下一处理器
        endTime = new Date();                            //记下结束时间
        duration = endTime.getTime()-startTime.getTime(); //计算所耗费时间
        System.out.println("本次请求/响应耗时："+duration+"毫秒!");
    }
    public void destroy() {
        this.filterConfig = null;
    }
}
```

web.xml 文件的主要内容如下：

```
<filter>
    <filter-name>pageTimers</filter-name>
    <filter-class>PageTimersFilter</filter-class>
</filter>
<filter-mapping>
    <filter-name>pageTimers</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

本实例的运行结果如图 7.4 所示。

```
Problems  Tasks  Web Browser  Console  Servers
tomcat6Server [Remote Java Application] C:\Program Files\Genuitec\Common\
信息: Starting service Catalina
2013-4-10 12:06:24 org.apache.catalina.core.StandardEn
信息: Starting Servlet Engine: Apache Tomcat/6.0.16
2013-4-10 12:06:25 org.apache.coyote.http11.Http11Prot
信息: Starting Coyote HTTP/1.1 on http-8080
2013-4-10 12:06:25 org.apache.jk.common.ChannelSocket
信息: JK: ajp13 listening on /0.0.0.0:8009
2013-4-10 12:06:25 org.apache.jk.server.JkMain start
信息: Jk running ID=0 time=0/31  config=null
2013-4-10 12:06:25 org.apache.catalina.startup.Catalin
信息: Server startup in 988 ms
本次请求/响应耗时: 15.0毫秒!
本次请求/响应耗时: 94.0毫秒!
```

图 7.4　使用 Filter 实现计时过滤器的运行结果

7.5　本章小结

本章主要介绍了什么是 Filter、Filter 的工作原理及主要功能，并且详细地讲解了 Filter 的配置文件。通过编码实例对 Filter 进行了系统的阐述，使读者进一步理解了 Filter，让读者掌握 Filter 中的主要方法和功能以及 Filter 的应用。

第8章

监 听 器

利用 Servlet 事件监听器(Listerner)可以完成一些非常有用的功能。例如,当发生一个 HttpSession 对象被创建的事件时,即可认为一个新用户进入本网站,在该事件处理程序中将当前在线统计人数加 1;当发生一个 HttpSession 对象被销毁的事件时,即可认为一个用户已经离开了本网站,在该事件处理程序中将当前在线统计人数减 1。

本章要点:

- 什么是 Listerner
- Listerner 的分类
- Listerner 的配置说明
- Listerner 的实例

8.1 Listener 简介

监听器是专门用于对其他对象身上发生的事件或状态改变进行监视和响应处理的对象,它好比现实生活中的盯梢者或侦探,当被监视的对象发生情况时,立即采用相应行动。Servlet 事件监听器是 Servlet 规范中定义的一种特殊的类,它用于监听 Web 应用程序中的 ServletContext、HttpSession 和 ServletRequest 等域对象创建与销毁的事件,以及监听这些域对象中的属性发生修改的事件。

按监听的对象划分,Servlet 2.4 规范中定义的事件监听器可以分为如下 3 种类型。

(1) 用于监听应用程序环境对象(ServletContext)的事件监听器。

(2) 用于监听用户会话对象(HttpSession)的事件监听器。

(3) 用于监听请求消息对象(ServletRequest)的事件监听器。

按照监听的事件类型进行划分,Servlet 2.4 规范中定义的事件监听器又可以分为如下 3 种类型。

(1) 用于监听域对象自身创建和销毁的事件监听器。

(2) 用于监听域对象中属性增加和删除的事件监听器。

(3) 用于监听绑定到 HttpSession 域中某个对象状态的事件监听器。

监听应用程序环境对象(ServletContext)的事件监听器需要使用如下两个接口类。

（1）ServletContextAttributeListener：监听对 ServletContext 属性的操作，如增加、删除、修改操作。

ServletContextAttributeListener 接口的方法如下：

```
void attributeAdded(ServletContextAttributeEvent scab);
//若有对象加入 Application 的范围，通知正在收听的对象
void attributeRemoved(ServletContextAttributeEvent scab);
//若有对象从 Application 的范围移除，通知正在收听的对象
void attributeReplaced(ServletContextAttributeEvent scab);
//若在 Application 的范围中，有对象取代另一个对象时，通知正在收听的对象
```

（2）ServletContextListener：监听 ServletContext，当创建 ServletContext 时，激发 contextInitialized（ServletContextEvent sce）方法；当销毁 ServletContext 时，激发 contextDestroyed(ServletContextEvent sce)方法。

监听用户会话对象（HttpSession）的事件监听器，可以监听 HTTP 会话活动情况、HTTP 会话中属性设置情况，也可以监听 HTTP 会话的 active、paasivate 情况等。该监听器需要使用到如下多个接口类。

（1）HttpSessionListener：监听 HttpSession 的操作。当创建一个 Session 时，激发 session Created（SessionEvent se）方法；当销毁一个 Session 时，激发 sessionDestroyed（HttpSessionEvent se）方法。

（2）HttpSessionActivationListener：用于监听 Http 会话 active、passivate 情况。

（3）HttpSessionAttributeListener：监听 HttpSession 中属性的操作。当在 Session 中增加一个属性时，激发 attributeAdded（HttpSessionBindingEvent se）方法；当在 Session 中删除一个属性时，激发 attributeRemoved(HttpSessionBindingEvent se)方法；在 Session 属性被重新设置时，激发 attributeReplaced（HttpSessionBindingEvent se）方法。

监听请求消息对象（ServletRequest）的事件监听器可以对客户端的请求进行监听，是在 Servlet 2.4 规范中新添加的一项技术，使用的接口类有 ServletRequestListener 和 ServletRequestAttributeListener。

在 Servlet 规范中为每种事件监听器都定义了相应的接口，在编写事件监听器程序时只需实现这些接口就可以了，Web 服务器根据用户编写的事件监听器所实现的接口把它注册到相应的被监听对象上。其中一些 Servlet 监听器需要在 Web 应用程序的部署描述符文件（web.xml 文件）中进行注册。一个 web.xml 文件中可以注册多个 Servlet 监听器，Web 服务器按照它们在 web.xml 文件中的注册顺序来加载和注册这些 Servlet 监听器。Servlet 事件监听器的注册和调用过程都是由 Web 容器自动完成的，当发生被监听的对象被创建、修改或销毁等事件时，Web 容器将调用与之相关的 Servlet 监听器对象的相应方法，用户在这些方法中编写的事件处理代码即被执行。在 web.xml 文件中 Listener 的配置代码如下：

```
<listener>
```

```
    <listener-class> MyServletContextListener</listener-class>
</listener>
```

8.2 Listener 的一般使用步骤

在 Java Web 应用中使用监听器的一般步骤如下。

(1) 编写一个实现某个 Listener 接口的监听器类,代码如下:

```
public class MyServletContextListener implements ServletContextListener, ServletCont
extAttributeListener{
    ...
}
```

(2) 实现监听器类中相应的方法,针对监听到的事件做出反应,代码如下:

```
public void contextInitialized(ServletContextEvent s) {
  //TODO 该方法实现了 ServletContextListener 接口定义的方法
  //对 ServletContext 进行初始化
  this.context = s.getServletContext();            //初始化一个 ServletContext 对象
  print("ServletContext 初始化...");                //打印出该方法的操作信息
}
```

(3) 将监听器类配置到 web.xml 文件中,代码如下:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4"
xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
    <listener>
        <listener-class>test.MyContextListener</listener-class>
    </listener>
</web-app>
```

8.3 Listener 的应用实例

【例 8.1】 两个监听器实例。在本实例中共创建两个 Listener 实例,分别用于监听 ServletContext、HttpSession 对象生命周期及 ServletContext、HttpSession 对象中属性变化情况。

(1) 创建一个用于监听 ServletContext 对象生命周期及 ServletContext 对象中属性变化情况的监听器 MyContextListener。MyContextListener 分别实现了 ServletContextListener 接口和 ServletAttributeListener 接口。

(2) 创建一个用于监听 HttpSession 对象生命周期及 HttpSession 对象中属性变化情况的监听器 MySessionListener。MySessionListener 分别实现了 HttpSessionListener 接口和

HttpSessionAttributeListener 接口。

(3) 在 web.xml 文件中配置 MyContextListener 和 MySessionListener，使之生效。

(4) 创建两个测试用的 JSP 页面 index.jsp 和 logout.jsp，在 JSP 页面中通过对 ServletContext、HttpSession 对象中属性的新增与调整来激发相应的过滤器。

MyContextListener.java 文件的代码如下：

```
import javax.servlet.ServletContextAttributeEvent;
import javax.servlet.ServletContextAttributeListener;
import javax.servlet.ServletContextEvent;
import javax.servlet.ServletContextListener;
public class MyContextListener implements ServletContextListener,
ServletContextAttributeListener {
    //当应用启动时将执行此方法
    public void contextInitialized(ServletContextEvent event) {
        System.out.println("【监听到】应用被启动!");
    }
    //当应用关闭时将执行此方法
    public void contextDestroyed(ServletContextEvent event) {
        System.out.println("【监听到】应用被关闭!");
    }
    //当 ServletContext 对象中新增属性时将执行此方法
    public void attributeAdded(ServletContextAttributeEvent event) {
        System.out.println("【监听到】ServletContext 对象中新增一名为"+event.getName()+
                            "的属性,其属性值为"+event.getValue());
    }
    //当 ServletContext 对象中删除属性时将执行此方法
    public void attributeRemoved(ServletContextAttributeEvent event) {
        System.out.println("【监听到】ServletContext 对象中一名为"+event.getName()+
                            "的属性被删除!");
    }
    //当 ServletContext 对象中某属性被更新时将执行此方法
    public void attributeReplaced(ServletContextAttributeEvent event) {
        System.out.println("【监听到】ServletContext 对象中一名为"+event.getName()+
                            "的属性被更新!");
    }
}
```

HttpSessionAttributeListener.java 文件的代码如下：

```
import javax.servlet.http.HttpSessionAttributeListener;
import javax.servlet.http.HttpSessionBindingEvent;
import javax.servlet.http.HttpSessionEvent;
import javax.servlet.http.HttpSessionListener;
public class MySessionListener implements HttpSessionListener,
HttpSessionAttributeListener {
    //在线人数计数器
    int userCount = 0;
```

```
    //当产生一个新的 HttpSession 对象(新用户上线)时执行此方法
    public void sessionCreated(HttpSessionEvent event) {
        //在线人数加 1
        event.getSession().getServletContext().setAttribute("count",new Integer(++userCount));
        System.out.println("【监听到】新用户"+event.getSession().getId()+"上线!");
    }
    //当一个 HttpSession 对象销毁(用户下线)时执行此方法
    public void sessionDestroyed(HttpSessionEvent event) {
        //人数减 1
        event.getSession().getServletContext().setAttribute("count",new Integer(--userCount));
        System.out.println("【监听到】用户"+event.getSession().getId()+"下线!");
    }
    //当 HttpSession 对象中新增属性时将执行此方法
    public void attributeAdded(HttpSessionBindingEvent event) {
        System.out.println("【监听到】HttpSession 对象中新增一名为"+event.getName()+
                              "的属性,其属性值为"+event.getValue());
    }
    //当 HttpSession 对象中删除属性时将执行此方法
    public void attributeRemoved(HttpSessionBindingEvent event) {
        System.out.println("【监听到】HttpSession 对象中一名为"+event.getName()+
                              "的属性被删除!");
    }
    //当 HttpSession 对象中更新属性时将执行此方法
    public void attributeReplaced(HttpSessionBindingEvent event) {
        System.out.println("【监听到】HttpSession 对象中一名为"+event.getName()+
                              "的属性被更新!");
    }
}
```

index.jsp 文件的代码如下：

```
<%@ page language="java" import="java.util.*" pageEncoding="gb2312"%>
<html>
<head>
<title>Listener 应用示例</title>
</head>
<body>
<h2>Listener 应用示例</h2>
<hr>
<%
    //在 application 中新增一属性
    application.setAttribute("admin","liubin");
    //在 application 中更新一属性
    application.setAttribute("admin","abcd");
    //在 application 中删除一属性
    application.removeAttribute("admin");
    //在 session 中新增一属性
    session.setAttribute("now",new Date());
```

```
        //在 session 中更新一属性
        session.setAttribute("now","2006 年 5 月 24 日");
        //在 session 中删除一属性
        session.removeAttribute("now");
    %>
    目前在线人数：<%=application.getAttribute("count")%>人!<br><br>
    <a href="logout.jsp">注销</a>
</body>
</html>
```

logout.jsp 文件的代码如下：

```
<%@ page language="java" import="java.util.*" pageEncoding="gb2312"%>
<html>
<head>
<title>Listener 应用示例</title>
</head>
<body>
<h2>Listener 应用示例</h2>
<hr>
<%
    //销毁 session 对象
    session.invalidate();
%>
恭喜您成功注销!<br><br>
目前在线人数：<%=application.getAttribute("count")%>人!
</body>
</html>
```

web.xml 文件的主要内容如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4"
    xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
    http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
    <listener>
        <listener-class> MyContextListener</listener-class>
    </listener>
    <listener>
        <listener-class> MySessionListener</listener-class>
    </listener>
</web-app>
```

本实例的运行效果如图 8.1～图 8.3 所示。

```
2013-4-11 10:29:07 org.apache.catalina.core.StandardContext addApplicationListener
信息: The listener "test.MySessionListener" is already configured for this context. The duplicate definition has been
【监听到】应用被启动!
2013-4-11 10:29:07 org.apache.coyote.http11.Http11Protocol start
信息: Starting Coyote HTTP/1.1 on http-8080
2013-4-11 10:29:07 org.apache.jk.common.ChannelSocket init
信息: JK: ajp13 listening on /0.0.0.0:8009
2013-4-11 10:29:07 org.apache.jk.server.JkMain start
信息: Jk running ID=0 time=0/62  config=null
2013-4-11 10:29:07 org.apache.catalina.startup.Catalina start
信息: Server startup in 2712 ms
【监听到】ServletContext对象中新增一名为org.apache.jasper.runtime.JspApplicationContextImpl的属性, 其属性值为org.apache
【监听到】ServletContext对象中新增一名为count的属性, 其属性值为1
【监听到】新用户A5214C6B83EA88F28D91625E3C77D764上线!
【监听到】ServletContext对象中新增一名为admin的属性, 其属性值为liubin
【监听到】ServletContext对象中一名为admin的属性被更新!
【监听到】ServletContext对象中一名为admin的属性被删除!
【监听到】HttpSession对象中新增一名为now的属性, 其属性值为Thu Apr 11 10:29:36 CST 2013
【监听到】HttpSession对象中一名为now的属性被更新!
【监听到】HttpSession对象中一名为now的属性被删除!
【监听到】ServletContext对象中一名为count的属性被更新!
【监听到】用户A5214C6B83EA88F28D91625E3C77D764下线!
```

图 8.1　运行结果控制台上输出的信息



图 8.2　index.jsp 页面效果

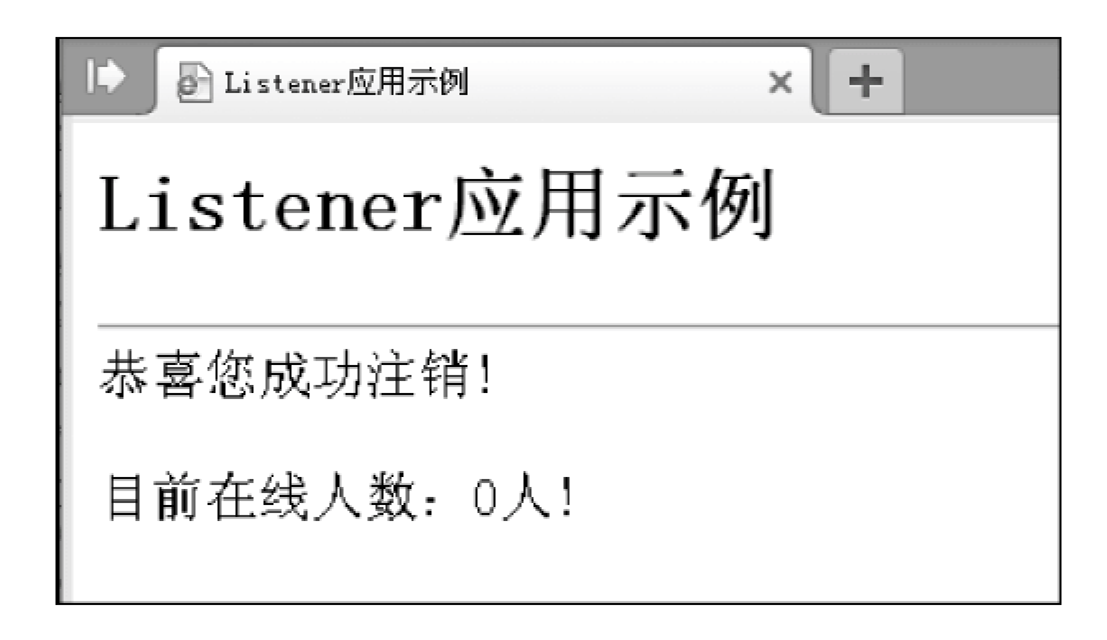


图 8.3　logout.jsp 页面效果

8.4　本章小结

本章主要介绍了什么是 Listener，Listener 的主要功能、接口和接口中的主要方法，并且讲解了 Listener 的配置文件。通过编码实例对 Listener 进行了系统的阐述，使读者进一步理解 Listener，同时在这一章中还讲解了读者要掌握 Listener 接口中的主要方法及功能以及 Listener 的应用。

第二篇

Struts 2

第9章

Struts 2 概述

当 MVC 设计模式在 Web 应用中大放异彩之后,各种基于 MVC 的框架应运而生,在如雨后春笋般涌现的众多框架中,Struts 2 框架以其诸多优点脱颖而出,获得了 Java 程序员的青睐。那么什么是框架? Struts 2 框架是如何产生的? 它有哪些优点? 它的体系架构包括什么? 下面我们就开始 Struts 2 框架的学习之路。

本章要点:

- 什么是框架
- Struts 2 框架的起源
- Struts 2 框架的优点
- Struts 2 框架的体系架构

9.1 什么是框架

框架(Framework),从应用方面定义,它是整个或部分系统的可重用设计,表现为一组抽象构件及构件实例间交互的方法;从目的方面定义,框架是可被应用开发人员定制的应用骨架。

可以说,一个框架是一个可复用的设计构件,它规定了应用的体系结构,阐明了整个设计、协作构件之间的依赖关系、责任分配和控制流程,表现为一组抽象类以及其实例之间协作的方法,它为构件复用提供了上下文(Context)关系。

应用框架可以提高软件的设计重用性和系统的可扩充性,缩短大型应用软件系统的开发周期,提高开发的质量。

9.2 Struts 2 起源

在框架的发展过程中,诞生了很多的设计思想,其中不乏优秀之辈,MVC 就是其中之一。MVC 将一个应用程序分成三个基本部分:模型(Model)、视图(View)和控制器(Controller)。三个部分以最少的耦合协同工作,从而提高应用的可扩展性和维护性。

视图是用户看到并与之交互的界面,例如 Web 应用中的网页。

模型表示企业数据和业务规则,例如实体类。

控制器接受用户的输入并调用模型和视图去完成用户的需求,例如 JSP 中

的 Servlet。

在 J2EE 平台上，MVC 是被广大 Java 开发人员所推崇的一种设计模式。在众多的基于 MVC 框架中，最具代表性的有三个框架：Struts、WebWork 和 Struts 2，其中，Struts 2 则是综合了前两者优点的优秀框架。

Struts 2 是一种全新的 MVC 框架技术，它比以往的所有其他 MVC 框架更加优秀。Struts 2 建立在 Struts 和 WebWork 基础之上，基于 WebWork 框架设计思想构建其核心部分，并加入了新的内容，使其更加适合当今 J2EE 开发的需求。

9.3 Struts 2 的优点

Struts 2 框架具备以下优点。

(1) 采用 MVC 思想，层次结构清晰，使程序员只需关注于业务逻辑部分的实现。

(2) 丰富的标签库，灵活运用 Struts 2 标签能大大提高开发效率。

(3) 优秀的拦截器：Struts 2 的拦截器是一个 Action 级别的 AOP，Struts 2 中的许多特性都是通过拦截器来实现的，例如异常处理、数据类型转换、验证等。拦截器是可配置与重用的，可以将一些通用的功能(如：登录验证、权限验证、记录日志等)设计到拦截器中以完成一些 Web 应用中比较通用的功能。

(4) 通过配置文件，就可以掌握整个系统各个部分之间的关系。

(5) 异常处理机制，只需在配置文件中配置异常的映射，即可对异常做相应的处理。

(6) 全局结果与声明式异常：为应用程序添加全局的 Result，和在配置文件中配置对异常进行处理，这样当处理过程中出现指定异常时，可以跳转到特定页面，这一功能十分实用。

(7) 使用 OGNL 进行参数传递：OGNL 提供了在 Struts 2 里访问各种作用域中数据的简单方式，可以方便地获取 Request、Attribute、Application、Session、Parameters 中的数据，大大简化了开发人员在获取这些数据时的代码量。

(8) 易于测试：Struts 2 的 Action 都是简单的 POJO，这样可以方便地对 Struts 2 的 Action 编写测试用例，大大方便了 Web 应用的测试。

(9) 支持 I18N。

(10) 开源框架：使开发人员能更深入的了解其内部实现机制。

9.4 Struts 2 的体系架构

9.4.1 Struts 2 的主要组成

Struts 2 框架主要由 3 个部分组成：核心控制器 FilterDispatcher、业务控制器和用户实现的业务逻辑组件。在这 3 个部分里，Struts 2 框架提供了核心控制器 FilterDispatcher，由用户实现业务控制器和业务逻辑组件。

Struts 2 采用 MVC 思想，其 MVC 模型各部分构成如下。

（1）控制器：Struts 2 框架核心控制器 FilterDispatcher，是一个 Servlet 过滤器，它运行在 Web 应用中，负责拦截所有的用户请求，当用户请求到达时，首先由 FilterDispatcher 过滤用户请求。如果用户请求以 Action 结尾（其实不以 Action 结尾也可以），该请求将被转入 Struts 2 框架处理，FilterDispatcher 决定由哪个 Action 来处理当前的请求。

（2）模型：Action 是 Struts 2 框架中的模型部分。Action 主要功能有两个：首先，Action 接收请求，然后调用业务逻辑来处理请求；其次，它还可以进行数据的传递。当业务控制器处理完用户请求后，根据处理结果不同，Action 的方法会返回不同的字符串——这个字符串是一个逻辑视图的名字——每一个字符串对应一个视图名。

（3）视图：对应视图组件，通常是指 JSP 页面，但也适用于其他视图显示技术，如 Velocity、FreeMarker、Excel、Tiles 等视图资源。当 Action 返回逻辑视图，视图组件调用对应的物理视图资源，并显示在客户端。

Struts 2 框架实现 MVC 架构如图 9.1 所示。

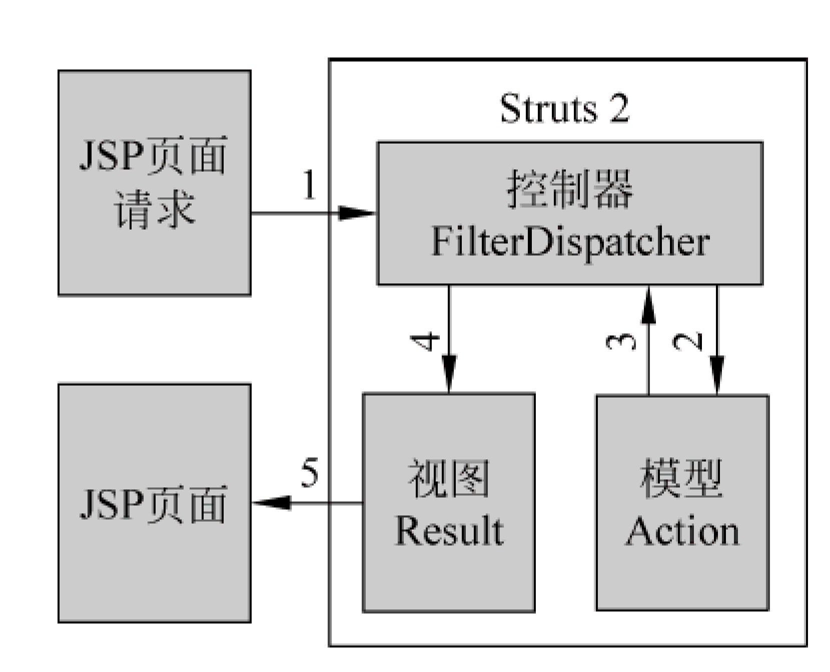


图 9.1　Struts 2 框架实现 MVC 架构

其执行流程如下。

（1）JSP 提交请求。

（2）FilterDispatcher 接收请求，并调用相应的 Action 处理请求。

（3）Action 调用业务逻辑组件处理业务逻辑，根据处理结果返回一个逻辑视图。

（4）FilterDispatcher 根据 Action 返回的逻辑视图，创建相应的物理视图。

（5）在页面显示物理视图。

9.4.2　Struts 2 工作流程

前面所讲的是 Struts 2 框架简单处理请求的过程，其完整的工作流程还有更多内容。Struts 2 官方所给出的工作流程图如图 9.2 所示。

这个流程图全面概括了一个请求从提交给 Struts 2 到最终生成响应并返回给客户端的全部流程，整个流程可以分为以下几个步骤。

（1）客户端初始化一个指向 Servlet 容器（例如 Tomcat）的请求——HttpServletRequest。

（2）这个请求经过一系列的过滤器（例如 ActionContextCleanUp）的过滤，传递给 FilterDispatcher。

（3）FilterDispatcher 接收到请求后，询问 ActionMapper 请求所对应的 Action 的映射信息。

（4）找到符合的映射信息后，FilterDispatcher 把请求的处理交给 ActionProxy。

（5）ActionProxy 通过 Configuration Manager 在框架的配置文件 struts.xml 中查找，找到需要调用的 Action 类。

（6）ActionProxy 创建一个被请求的 Action 的实例，该实例用于处理请求信息。

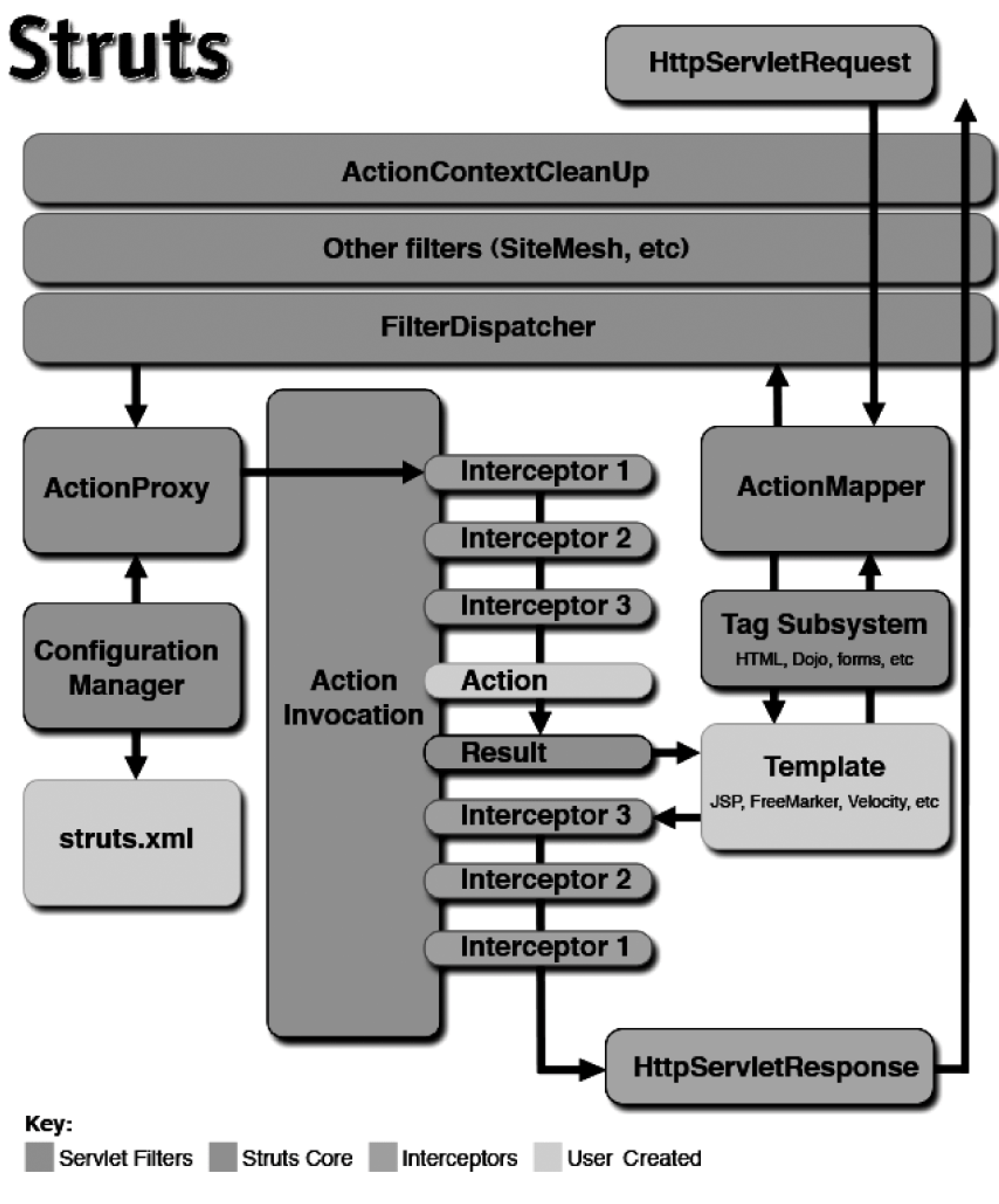


图 9.2　Struts 2 工作流程图

(7) 如果在 struts.xml 配置文件中存在与被请求 Action 相关的拦截器配置，那么在此 Action 实例被调用的前后，这些拦截器 Interceptor 会依次被执行。

(8) Action 执行完毕，会返回一个逻辑视图，ActionInvocation 负责根据 struts.xml 中配置的 Result 找到对应的物理视图。这个物理视图通常是一个 JSP，也可能是另外一个 Action 链或者 FreeMarker 模板、Velocity 模板，等等。

(9) 物理视图被返回给客户端。

在上面的描述中，图示中的元素都使用双引号括起来，以方便大家在图中查找其对应的位置。

另外，由于图 9.2 是 Struts 2 框架的整个工作流程图，在还未开始应用 Struts 2 框架之前，大家并不容易理解，建议大家在以后的学习中，逐步消化、理解。本书在各章节也会勾画图示中相应的部分，随着学习的深入，逐步填充至完整；当全书学习完成以后，就可以透彻地理解整个流程。

9.5 本章小结

本章的主要目的是引导读者对 Struts 框架进行初步的了解和整体的认识，先讲解了什么是框架、应用框架的好处是什么；接着引入 Struts 2 框架的起源，讲解了其优点和体系架构。

第10章

构建第一个 Struts 2 应用

在对 Struts 2 框架有了初步认识后，本章首先学习如何构建一个 Struts 2 框架的 Web 应用(以后简称为 Struts 2 应用)。要构建 Struts 2 应用，首先需要安装和配置相关的开发环境，包括 Web 容器(本书使用 Tomcat)、Java 开发环境及 Struts 2 框架运行所必需的类库等。本章将介绍两种方式来构建一个 Struts 2 应用：完全手工搭建和使用 MyEclipse 搭建。

本章要点：

- Struts 2 框架最基本的构成
- Struts 2 框架基本原理
- Struts 2 框架的类库
- 搭建最简单 Struts 2 应用的过程和方法

10.1 增加 Struts 框架前的准备工作

在学习使用 Struts 2 开发应用程序之前，需要先准备好相应的开发环境，包括 Java 开发环境(JDK 1.6)、Web 容器(Tomcat 6.0)及 Struts 2 框架的资源包。

下载并安装 JDK 和 Tomcat 在前面的章节中已讲过，此处不再重复。但需要注意的是，为防止版本不兼容导致程序错误，JDK 要求使用 JDK 1.6 版本，Tomcat 要求使用 Tomcat 6.0 版本。

Struts 2 框架的资源包可以在其官方网站下载，网址是 http://struts.apache.org/download。下载的选项分为如下几个选项。

(1) Full Distribution：下载 Struts 2 的完整版。通常建议下载该选项，它包含接下来 6 个选项中的全部内容。

(2) Example Applications：下载 Struts 2 的示例应用，这些示例应用对于学习 Struts 2 有很大的帮助。

(3) Blank Application only：仅下载 Struts 2 的空示例应用，这个空应用已经包含在 Example Applications 选项下。

(4) Essential Dependencies：仅下载 Struts 2 的核心库。

(5) Documentation：仅下载 Struts 2 的相关文档，包含 Struts 2 的使用文档、参考手册和 API 文档等。

(6) Source：下载 Struts 2 的全部源代码。

通常建议读者选择第一个选项——下载 struts2-2.2.1-all.zip，这个压缩包是 Sturts 2 的完整资源包。将下载到的压缩包解压，得到 struts-2.2.1 文件夹，该文件夹包含如下文件结构。

(1) apps：该文件夹下包含了基于 Struts 2 的示例应用，这些示例应用对于学习者是非常有用的资料。

(2) docs：该文件夹下包含了 Struts 2 的相关文档，包括 Struts 2 的快速入门、Struts 2 的文档，以及 API 文档等内容。

(3) lib：该文件夹下包含了 Struts 2 框架的核心类库，以及 Struts 2 的第三方插件类库。

(4) src：该文件夹下包含了 Struts 2 框架的全部源代码。

提示：Struts 2 框架需要依赖于 XWork，因此读者在阅读 Struts 2 源代码时需要结合 XWork 的源码，而在 Struts 2 资源包的 src 文件夹下并没有提供 XWork 的源代码，所以读者需要自行到 www.opensymphony.com/xwork 站点下载 XWork 的源代码。

10.2　手动搭建 Struts 2 应用程序

虽然手动搭建应用程序不如使用 IDE 工具容易，但能帮助我们更好地了解程序的架构。笔者一直相信，要想成为优秀的程序员，扎实的基本功是一切的契机，所有的代码都可以使用简单的文本编辑器(包括 EditPlus、UltraEdit 等工具)完成。优秀的程序员当然可以、也应该使用 IDE 工具，但绝不能离不开 IDE 工具。当你真正掌握这门技术的各种细节后，就可以得心应手地使用任何一种 IDE 工具了。

在本节中，我们手动搭建第一个 Struts 应用——论坛管理系统。该应用实现的功能非常简单，当用户提交"进入论坛管理系统后台登录页面"的请求时，显示后台用户登录页面。我们可以通过以下几个步骤来实现这个最简单的应用。

(1) 搭建一个 Web 应用程序框架结构，这个框架结构也是应用 Struts 2 框架的结构。

(2) 加载 Struts 2 核心类库。

(3) 创建并配置 web.xml 文件。

(4) 创建并编写视图文件 login.jsp。

(5) 创建并配置 struts.xml 文件。

(6) 测试。

10.2.1　搭建 Struts 2 应用程序框架结构

Struts 2 应用就是一个普通的 Web 应用，增加了 Struts 2 功能，该应用就可以利用 Struts 2 实现 MVC 架构了。以后，我们将应用 Struts 2 框架的 Web 应用程序工程简称为 Struts 2 工程。

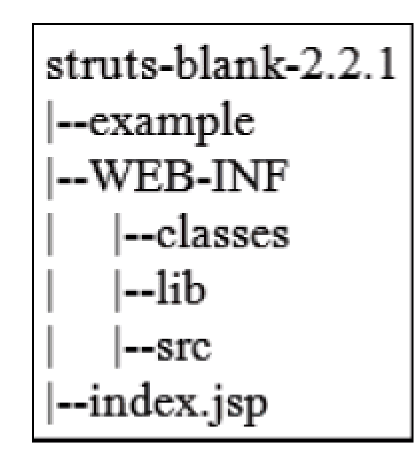

struts-blank-2.2.1
|--example
|--WEB-INF
|　|--classes
|　|--lib
|　|--src
|--index.jsp

图 10.1　文件目录结构

在搭建之前，我们要先了解 Struts 2 应用的框架结构。Struts 2 资源包的 apps 文件夹下有 Struts 2 应用的一些示例，可以供我们参考。其中，struts2-blank-2.2.1.war 文件，是一个最基本的 Struts 2 工程。我们可以把这个工程发布到 Tomcat 上，以了解 Struts 2 工程的文件目录结构。通过分析，最终我们得到 struts2-blank-2.2.1 工程的基本文件目录结构如图 10.1 所示。

图 10.1 中各文件夹的作用如表 10.1 所示。

表 10.1　文件夹的作用

配置元素名称	功 能 描 述
example	放置工程中所有的 JSP 文件
WEB-INF	放置 classes、lib、src 文件夹和 web.xml 文件
classes	放置所有编译后的 class 文件及各个配置文件
lib	放置工程运行所需的类库文件
src	放置 Java 源码文件和 struts.xml 文件

根据以上分析，创建论坛管理系统的框架结构如下。

(1) 在任意目录创建一个文件夹名为 Struts2_0100_first，该文件夹用于创建 Web 应用。

(2) 在第(1)步所创建的文件夹内，创建名为 WEB-INF 的文件夹。

(3) 进入 Tomcat，找到任何一个 Web 应用(例如 Tomcat 根目录的 webapps 文件夹下的 examples)，将 Web 应用的 WEB-INF 的文件夹下的 web.xml 文件复制到第(2)步所建的 WEB-INF 文件夹下。

(4) 修改复制的 web.xml 文件，将其改成只有一个根元素的 XML 文件，修改后的 web.xml 文件代码如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
</web-app>
```

(5) 在第(2)步所创建的 WEB-INF 文件夹下，再创建两个文件夹，名为 classes 和 lib。

经过以上步骤，已经建立了一个空的 Web 应用，将该 Web 应用复制到 Tomcat 的 webapps 文件夹下，该 Web 应用将可以自动部署在 Tomcat 中。

(6) 为方便管理后台 JSP 文件，在 Struts2_0100_first 文件夹下，创建 main 文件夹，

用于放置后台 JSP 文件。

(7) 在 main 文件夹下，创建 login.jsp，输入以下代码。

```
<%@ page contentType="text/html; charset=UTF-8" %>
<html>
<head>
<title>login</title>
</head>
<body>
    这是登录页面。
</body>
</html>
```

启动 Tomcat，在地址栏中输入 login.jsp 文件的地址——http://localhost:8080/Struts2_0100_first/login.jsp，浏览 login.jsp 文件，效果如图 10.2 所示。

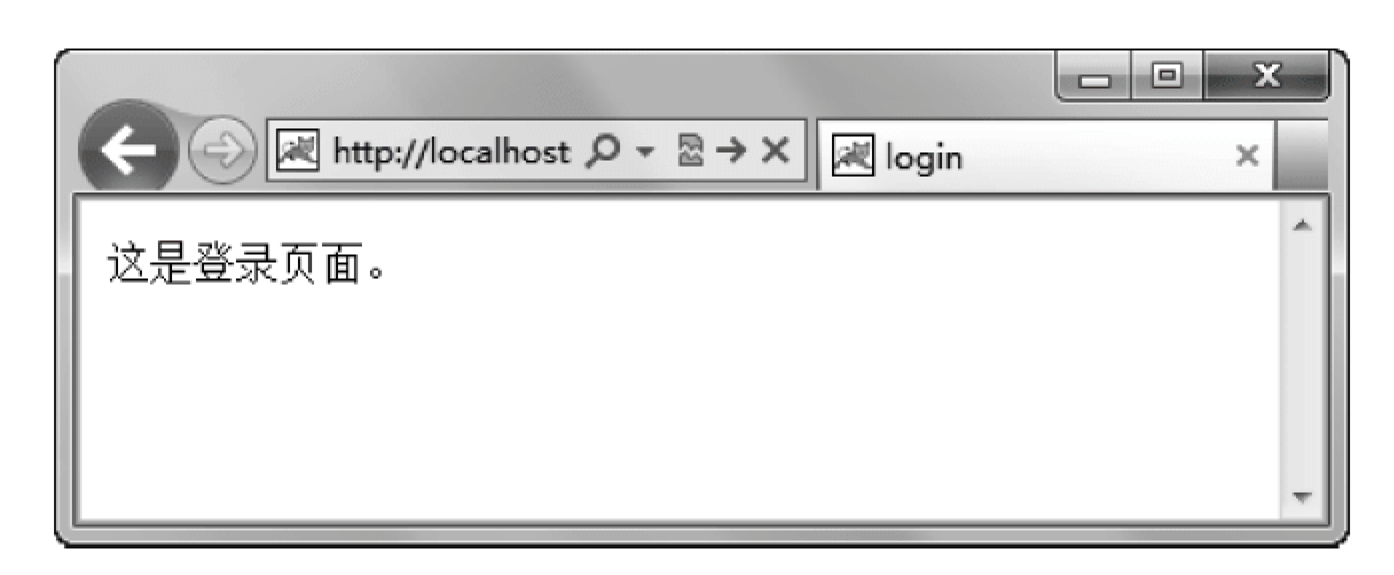


图 10.2　登录界面

10.2.2　增加 Struts 2 支持

要为 Web 应用增加 Struts 2 支持功能，只须将 Struts 2 安装到 Web 应用中即可，其步骤如下。

(1) 将 Struts 2 框架的常用类库复制到 Web 应用的 WEB-INF/lib 路径下。

常用类库文件包括：commons-fileupload-1.2.1.jar、commons-io-1.3.2.jar、freemarker-2.3.16.jar、javassist-3.7.ga.jar、ognl-3.0.jar、struts2-core-2.2.1.jar 和 xwork-core-2.2.1.jar。

这些类库文件在 Struts 2 资源包的 lib 文件夹下可以找到，在工程 struts2-blank-2.2.1 的 lib 文件夹下也可以找到。

(2) 修改 web.xml，在 web.xml 中配置 Struts 2 的核心控制器，代码如下：

```
<filter>
    <filter-name>struts2</filter-name>
    <filter-class>
        org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter
    </filter-class>
</filter>
```

```
<filter-mapping>
    <filter-name>struts2</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

其中，StrutsPrepareAndExecuteFilter 是 Struts 2 的核心控制器(即图 9.2 所示的 FilterDispatcher)，它被设计成了过滤器，因此需要通过<filter></filter>标记引入。

<url-pattern>/*</url-pattern>表示所有的客户端请求都经由 StrutsPrepareAndExecuteFilter 处理。

StrutsPrepareAndExecuteFilter 的功能包括拦截所有客户端的请求，并把过滤后的请求交给 Struts 2 进行处理。

10.2.3 创建并配置 struts.xml

Struts 框架的核心配置文件是 struts.xml。

1. 创建 struts.xml

在示例工程 struts2-blank-2.2.1 的 WEB-INF\src\java 路径下，找到 struts.xml，复制到 Struts2_0100_first/WEB-INF/lib 路径下。

修改其代码如下所示。

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
    "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
</struts>
```

这是一个空的配置文件。

2. 配置 struts.xml

在<struts>和</struts>标签中输入以下代码。

```
<package name="default" namespace="/main" extends="struts-default">
    <action name="login">
        <result>
            /main/login.jsp
        </result>
    </action>
</package>
```

以上代码配置请求与视图的对应关系。

启动 Tomcat，在浏览器中输入请求 http://localhost:8080/Struts2_0100_first/main/login，界面显示登录页面，效果如图 10.2 所示。

观察 Struts 2 的配置文件代码，定义了 Action 元素。在定义该 Action 时，定义了它

的 name 属性和 result 子元素，其中 name 属性决定了该 Action 处理用户的哪个请求，result 子元素定义对应的物理视图。

反过来说，以我们创建的 Web 应用为例，在请求 http://localhost:8080/Struts2_0100_first/main/login 中，Struts2_0100_first 是 Web 应用的工程名，"/main"与 package 元素中 namespace="/main"相对应，"login"与 action 元素中 name="login"相对应，显示在浏览器中的页面为<result>和</result>之间的"/main/login.jsp"所指定的页面。

这里我们学到了 Struts 2 的一个功能，就是将请求与视图分开。将请求与视图分开使程序更加灵活，当要更换视图时，只需要修改配置文件中<result>和</result>标签之间的视图文件路径即可。

10.3　使用 MyEclipse 创建 Struts 2 应用

通过 10.2 节的学习，我们对 Struts 2 工程架构有了足够的了解，下面开始学习使用 IDE 工具开发。使用 IDE 工具可以极大地提高开发效率。本节以目前最流行的 MyEclipse 为例，讲解使用 IDE 工具开发 Struts 2 应用程序的流程。

在本节中，我们使用 MyEclipse 同样搭建一个"论坛管理系统"的 Struts 应用。该应用非常简单，当用户提交"进入论坛管理系统后台登录页面"的请求时，显示后台用户登录页面。我们可以通过以下几个步骤来实现这个最简单的应用。

(1) 使用 MyEclipse 创建 Web 应用，配置 JDK 和 Tomcat。

(2) 加载 Struts 2 核心类库。

(3) 修改配置文件 web.xml。

(4) 创建并编写视图文件 login.jsp。

(5) 创建并配置 struts.xml 文件。

(6) 测试。

10.3.1　开发环境的准备

首先准备开发所需的软件环境，包括 JDK 1.6、Tomcat 6.0 和 MyEclipse 8.0。

这里所提供的 JDK、Tomcat 是与 Struts 2.2 相应的软件环境，为避免版本不同而导致的处理细节差异或产生版本不兼容引起的错误等，请读者做实验时，也采用以上版本。MyEclipse 本书采用了 8.0 的版本，读者采用其他版本也同样适用。

10.3.2　创建 Web 应用

打开 MyEclipse，创建 Web 应用，名为 Struts2_0100_Introduction，并配置 Tomcat 6.0 和 JDK 1.6。注意在创建工程时，选择 J2EE specification Level 时选择 Java EE 5.0，如图 10.3 所示。

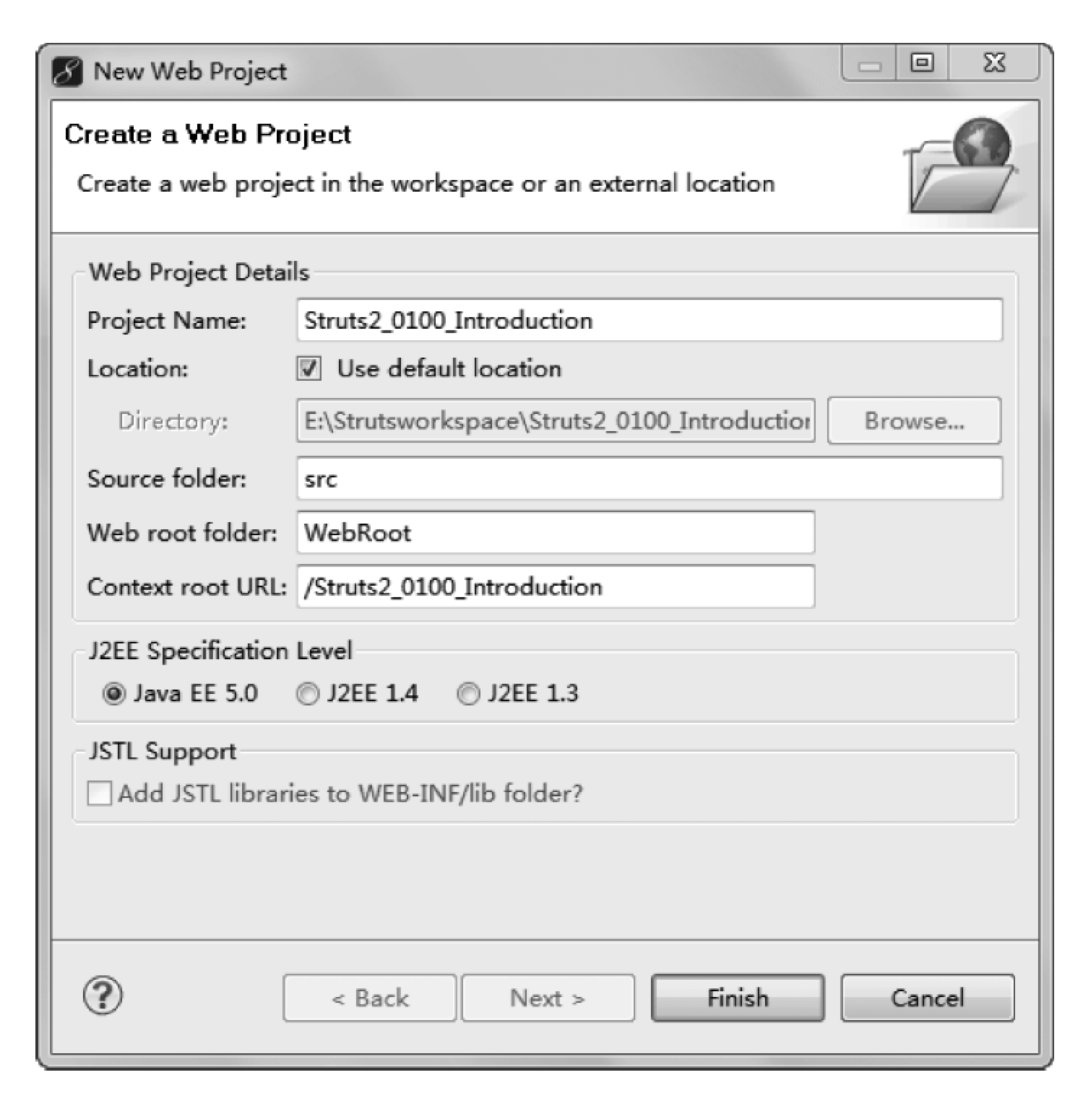


图 10.3 创建 Web 应用

10.3.3 增加 Struts 2 支持

要让 Web 应用具有 Struts 2 支持功能，需要将 Struts 2 框架核心类库加载到 Web 应用中，并将请求交给 Struts 处理，操作步骤如下。

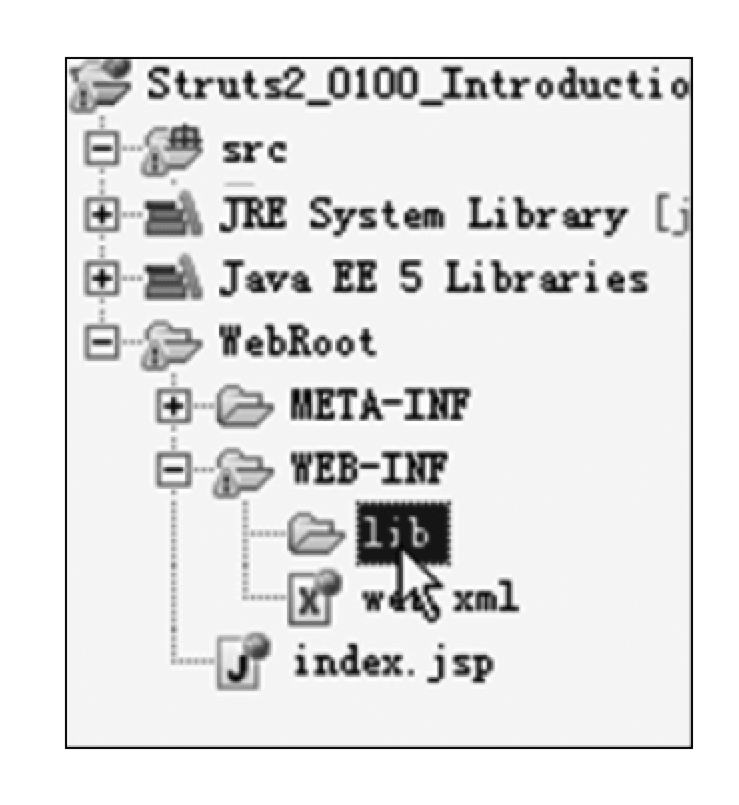


图 10.4 增加 Struts 2 支持

(1) 加载核心类库。在 Struts 2 框架的 lib 路径下找到常用类库文件：commons-fileupload-1.2.1.jar、commons-io-1.3.2.jar、freemarker-2.3.16.jar、javassist-3.7.ga.jar、ognl-3.0.jar、struts2-core-2.2.1.jar 和 xwork-core-2.2.1.jar。或在 Struts 2 框架的示例工程 struts2-blank-2.2.1 的 lib 文件夹下选中所有的 jar 包文件，即以上 7 个文件。将核心类库复制，并打开 MyEclipse 粘贴到 Web 应用的 lib 路径下，如图 10.4 所示。

(2) 修改 web.xml，在 web.xml 中配置 Struts 2 的核心控制器。

因为 Struts 2 的核心控制器 StrutsPrepareAndExecuteFilter 被设计成了过滤器，因此使用<filter></filter>标签配置，代码如下：

```
<filter>
    <filter-name>struts2</filter-name>
```

```
    <filter-class>
        org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter
    </filter-class>
</filter>
<filter-mapping>
    <filter-name>struts2</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

其中，<url-pattern>/*</url-pattern>表示所有的客户端请求都经由 StrutsPrepareAndExecuteFilter 处理，并把过滤后的请求交给 Struts 2 进行处理。

Struts 2 框架是开源框架，我们可以查看其核心控制器 StrutsPrepareAndExecuteFilter 的代码，方法如下。

（1）StrutsPrepareAndExecuteFilter 在 struts2-core-2.2.1.jar 中，因此在工程的 struts2-core-2.2.1.jar 处右击，在弹出的菜单中选择 Properties 命令，如图 10.5 所示。

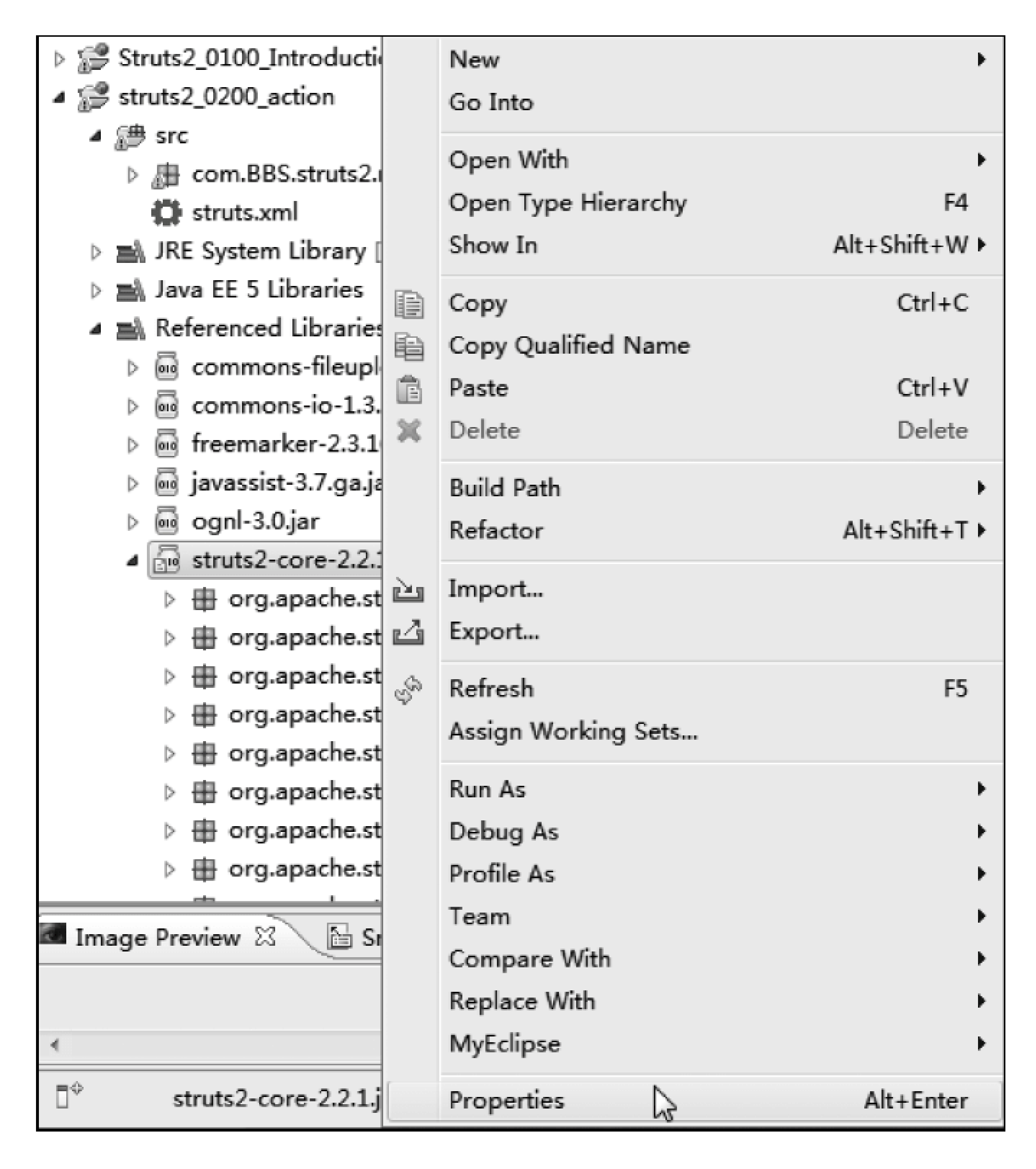


图 10.5 选择 Properties 命令

（2）在弹出的对话框中，选择 Java Source Attachment 选项，单击右侧的 External Folder 按钮，选择 Struts 2 框架资源包文件夹下/src/core 路径，笔者的资源包文件夹 struts-2.2.1 放在了 D 盘的根目录中，则路径为 D:/struts-2.2.1/src/core，如图 10.6 所示。

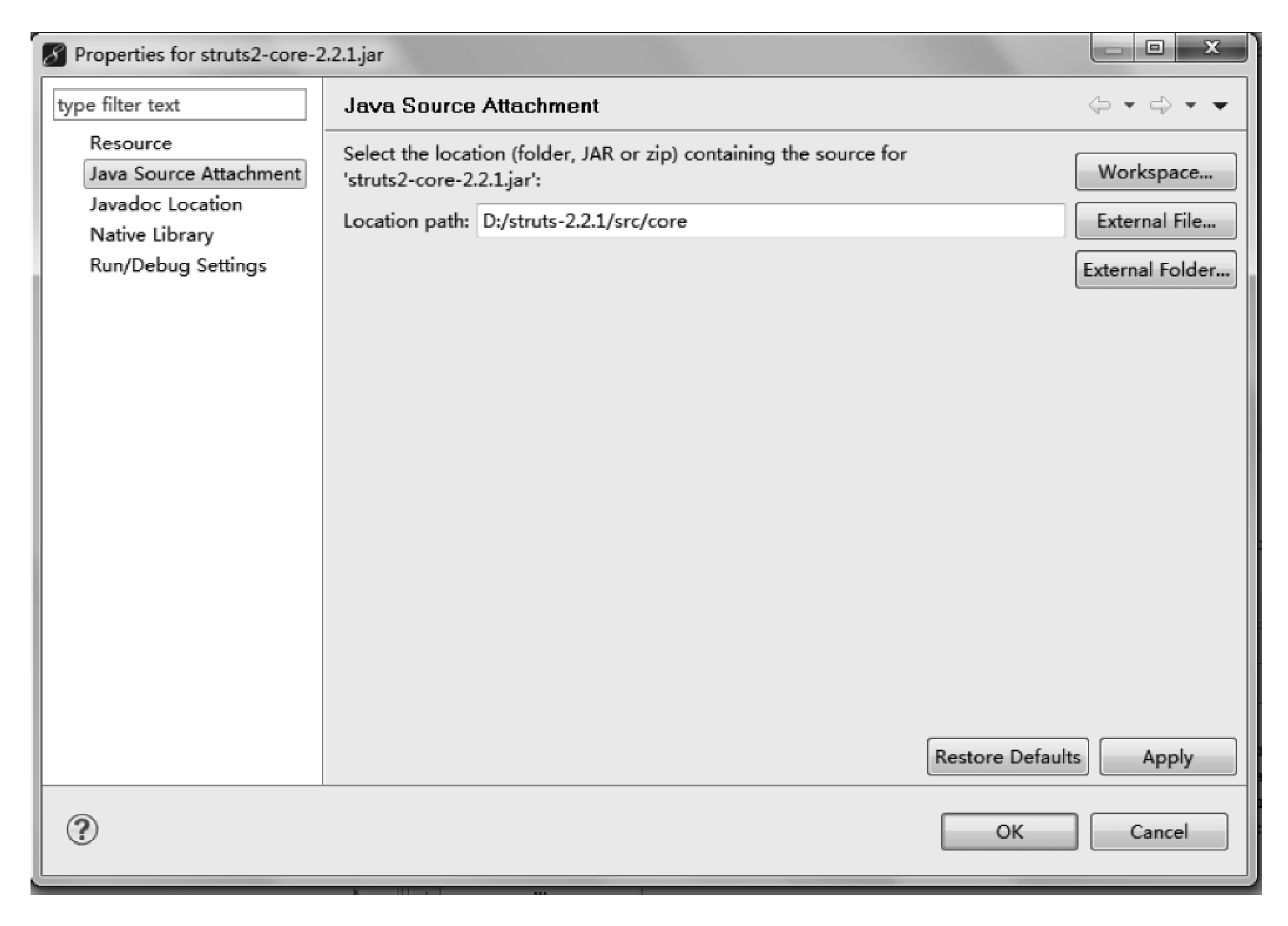


图 10.6 指定源代码路径

(3) 单击 OK 按钮，确定设置。

(4) 在工程中，打开 struts2-core-2.2.1.jar，在 org.apache.struts2.dispatcher.ng.filter 包中双击 StrutsPrepareAndExecuteFilter.class，就可以看到其源代码，如图 10.7 所示。

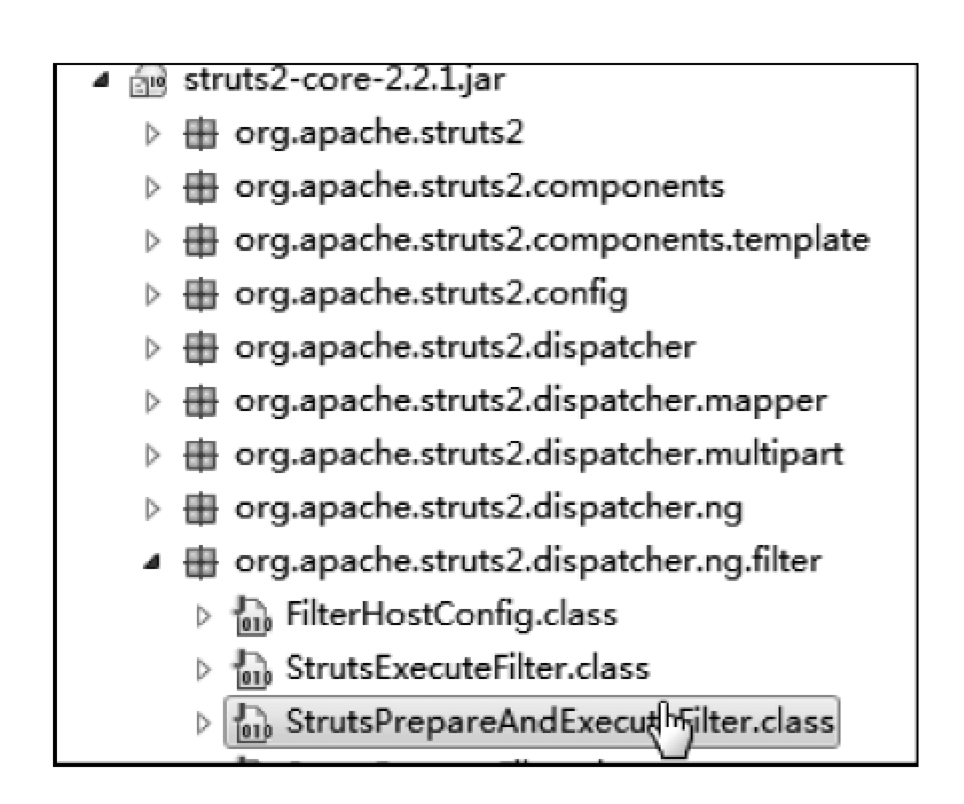


图 10.7 双击 StrutsPrepareAndExecuteFilter.class

其源代码如下：

```
package org.apache.struts2.dispatcher.ng.filter;

import org.apache.struts2.StrutsStatics;
import org.apache.struts2.dispatcher.Dispatcher;
```

```
import org.apache.struts2.dispatcher.mapper.ActionMapping;
import org.apache.struts2.dispatcher.ng.ExecuteOperations;
import org.apache.struts2.dispatcher.ng.InitOperations;
import org.apache.struts2.dispatcher.ng.PrepareOperations;

import javax.servlet.*;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.util.List;
import java.util.regex.Pattern;

/**
 * Handles both the preparation and execution phases of the Struts dispatching process.  This
   filter is better to use
 * when you don't have another filter that needs access to action context information, such as
   Sitemesh.
 */
public class StrutsPrepareAndExecuteFilter implements StrutsStatics, Filter {
    protected PrepareOperations prepare;
    protected ExecuteOperations execute;
    protected List<Pattern> excludedPatterns = null;

    public void init(FilterConfig filterConfig) throws ServletException {
        InitOperations init = new InitOperations();
        try {
            FilterHostConfig config = new FilterHostConfig(filterConfig);
            init.initLogging(config);
            Dispatcher dispatcher = init.initDispatcher(config);
            init.initStaticContentLoader(config, dispatcher);

            prepare = new PrepareOperations(filterConfig.getServletContext(), dispatcher);
            execute = new ExecuteOperations(filterConfig.getServletContext(), dispatcher);
            this.excludedPatterns = init.buildExcludedPatternsList(dispatcher);

            postInit(dispatcher, filterConfig);
        } finally {
            init.cleanup();
        }

    }

    /**
     * Callback for post initialization
     */
    protected void postInit(Dispatcher dispatcher, FilterConfig filterConfig) {
    }

    public void doFilter(ServletRequest req, ServletResponse res, FilterChain chain)
```

```
        throws IOException, ServletException {

            HttpServletRequest request = (HttpServletRequest) req;
            HttpServletResponse response = (HttpServletResponse) res;

            try {
                prepare.setEncodingAndLocale(request, response);
                prepare.createActionContext(request, response);
                prepare.assignDispatcherToThread();
                if (excludedPatterns != null &&
                prepare.isUrlExcluded(request, excludedPatterns)) {
                    chain.doFilter(request, response);
                } else {
                    request = prepare.wrapRequest(request);
                    ActionMapping mapping = prepare.findActionMapping(request, response, true);
                    if (mapping == null) {
                        boolean handled = execute.executeStaticResourceRequest(request, response);
                        if (!handled) {
                            chain.doFilter(request, response);
                        }
                    } else {
                        execute.executeAction(request, response, mapping);
                    }
                }
            } finally {
                prepare.cleanupRequest(request);
            }
        }

        public void destroy() {
            prepare.cleanupDispatcher();
        }
    }
```

我们可以看到，StrutsPrepareAndExecuteFilter 是一个过滤器，有 init、doFilter 和 destroy 等方法。

10.3.4 创建并配置 struts.xml

Struts 框架的核心配置文件是 struts.xml。

1. 创建 struts.xml

在示例工程 struts2-blank-2.2.1 的 WEB-INF\src\java 路径下，找到 struts.xml，复制到 Struts2_0100_first/WEB-INF/lib 路径下。

修改其代码如下所示。

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
```

```
    "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
    "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
</struts>
```

这是一个空的配置文件。

2. 配置 struts. xml

在<struts>和</struts>标签中输入以下代码。

```
<package name="default" namespace="/main" extends="struts-default">
    <action name="login">
        <result>
            /main/login.jsp
        </result>
    </action>
</package>
```

以上代码配置请求与视图的对应关系。

因为工程中并没有 main 文件夹和 login. jsp 文件，因此，将工程中 WebRoot 下的 index. jsp 文件改名为 login. jsp，并在 WebRoot 路径下创建文件夹 main，并将 login. jsp 文件移动到 main 文件夹下。

修改网页 pageEncoding 为 UTF-8，修改网页标题为 Login，修改页面显示内容为“这是登录页面”，其主要代码如下：

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
...
<html>
  <head>
    ...
    <title>Login</title>
    ...
  </head>

  <body>
    这是登录页面。<br/>
  </body>
</html>
```

启动 Tomcat，在浏览器中输入请求 http://localhost:8080/Struts2_0100_Introduction/main/login，界面显示登录页面，效果如图 10.2 所示。

观察 struts. xml 配置文件代码，在<struts>和</struts>标签之间定义了 package、action 和 result。下面我们分别解释一下这三个标签。

(1) 在 Struts 2 框架中，action、result 等配置信息是通过包(package)来管理的。包的 namespace 属性设置包的命名空间，它的作用是对 action 进一步的组织和划分，这样就可以在不同的 package 中，定义 name 值相同的 action。

(2) <action>标签的 name 属性指定该 Action 负责处理的请求 URL。例如在 Web

应用 Struts2_0100_Introduction 的配置文件中,<action>的 name 值为 login,表示该 Action 负责处理向 login 的 URL 发送的请求。

(3) <result../>定义 Action 的处理结果,指定 Action 逻辑视图与物理视图之间的映射关系。<result>/main/login.jsp</result>表示物理视图的路径是 Web 应用根目录下 main 文件夹下的 login.jsp。

反过来说,在请求 http://localhost:8080/Struts2_0100_Introduction/main/login 中,Struts2_0100_Introduction 为 Web 应用的工程名,/main 与 package 元素中 namespace="/main"相对应,login 与 action 元素中 name="login"相对应,显示在浏览器中的页面为<result>和</result>之间的/main/login.jsp 所指定的页面。

其中,package 元素中 namespace 的属性值和 action 元素中 name 的属性值可以任意,当其属性值发生变化时,只需要修改请求的相应部分。当然,属性值的名字最好可以顾名思义。

Struts 2 的一个重要功能,就是将请求与视图分开。将请求与视图分开使程序更加灵活,例如想要更换视图时,只需要修改配置文件中<result>和</result>标签之间的视图文件路径即可。

10.4 最简单 Struts 2 应用的处理请求流程

前两节我们分别使用手动和 IDE 工具,搭建了最简单的 Struts 2 应用,并演示了 Struts 2 如何将请求与视图分开。我们也了解到 StrutsPrepareAndExecuteFilter 作为过滤器运行的 Web 应用中,负责拦截所有用户的请求,然后将该请求转入 Struts 框架处理。那么,这个最简单的 Struts 2 应用是如何处理请求的呢?

下面是 Struts 2 处理请求最简单的流程。

(1) 加载核心控制器:org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter。

(2) 读取配置:struts 配置文件 struts.xml 中未指定 Action,则 Struts 2 框架自动创建一个 Action 类对象。

(3) 派发请求:用户提交请求 http://localhost:8080/Struts2_0100_Intorduction/main/login,Servlet 读取配置文件 web.xml,将请求交给过滤器 StrutsPrepareAndExecuteFilter。

(4) 调用 Action:从 struts.xml 文件中读取与 login 对应的 Action,因为 action 标签并未指定 Action,则调用 Struts 2 框架自动创建 Action。

(5) 查找响应:根据<result>/main/login.jsp</result>,跳转到根目录下 main 文件夹下的 login.jsp 页面。

(6) 响应用户:客户浏览器端显示 login.jsp。

其中第(1)步和第(2)步在部署工程、启动 Tomcat 时执行。其他步骤,读者可参看图 9.1 所示的流程,帮助理解。

第(4)步中所提到的 Action 将在第 11 章中详细解释,如果读者觉得理解困难,可以

暂时忽略,待学习完第 11 章就可以轻松理解了。

为了避免过多繁杂的流程给读者造成困扰,在以上流程中,有一些流程略去了,并未写出,将在以后的章节中逐步补充。

接下来,我们一起做几个实践任务,以此熟练操作和进一步理解流程。

10.5　实践任务 1:搭建简单的 Struts 2 应用

1. 任务说明

本任务以论坛管理系统的前台用户管理模块为例,实践搭建 Struts 2 应用,分离请求与视图。当用户提交“进入论坛管理系统前台登录页面”的请求时,显示前台用户登录页面。

当用户输入请求 http://localhost:8080/struts2_0110_login/login,浏览器打开前台登录页面,如图 10.8 所示。

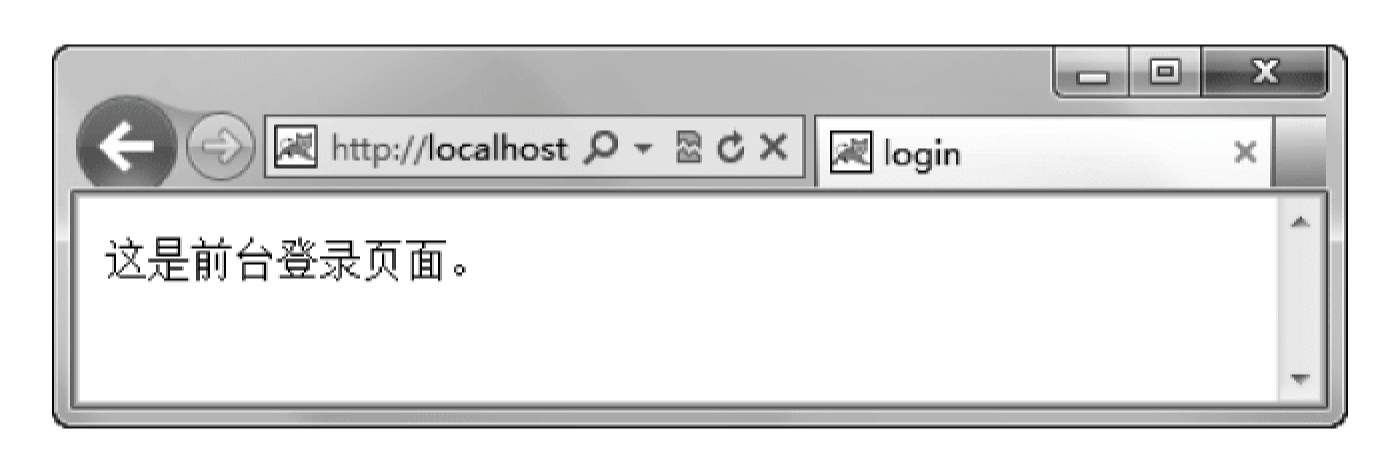


图 10.8　前台登录页面

2. 任务实施

(1) 创建 Web 应用。使用 MyEclipse 创建 Web 应用,名为 struts2_0110_login,配置 Tomcat 6.0 和 JDK 1.6。

(2) 创建视图层 jsp 页面。将 WebRoot 路径下的 index.jsp 文件,改名为 login.jsp,编写主要代码如下:

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
...
<html>
  <head>
    ...
    <title>Login</title>
    ...
  </head>

  <body>
    这是前台登录页面。<br/>
  </body>
</html>
```

(3) 增加 Struts 2 支持。

① 加载 Struts 2 核心类库。打开 Struts 2 资源包的 apps 文件夹,将示例工程

struts2-blank-2.2.1.war 文件解压,得到 struts2-blank-2.2.1 文件夹。

复制 struts2-blank-2.2.1\WEB-INF\lib 路径下的 7 个 jar 文件(即 Struts 2 框架的常用核心类库),粘贴到 struts2_0110_login 工程的 lib 路径下,如图 10.9 所示。

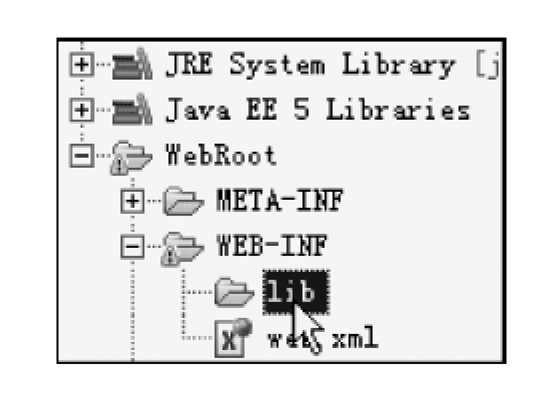


图 10.9　加载 Struts 2 核心类库

② 修改配置文件 web.xml。修改 web.xml 代码,将 Struts 2 框架的核心控制器配置为过滤器,过滤用户提交的所有请求。代码如下:

```
<filter>
    <filter-name>struts2</filter-name>
    <filter-class>org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter
    </filter-class>
</filter>

<filter-mapping>
   <filter-name>struts2</filter-name>
   <url-pattern>/*</url-pattern>
</filter-mapping>
```

(4) 创建配置文件 struts.xml。打开示例工程 struts2-blank-2.2.1 的 WEB-INF\src\java 路径,将 struts.xml 文件复制到 struts2_0110_login 的 src 路径下,并清空原来的配置代码,修改后的代码如下:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
    "http://struts.apache.org/dtds/struts-2.0.dtd">

<struts>
</struts>
```

(5) 修改配置文件 struts.xml。在<struts>和</struts>标签中输入以下代码。

```
<package name="default" namespace="/" extends="struts-default">
    <action name="login">
        <result>
            /login.jsp
        </result>
    </action>
</package>
```

(6) 测试。部署工程,在浏览器中输入请求代码 http://localhost:8080/struts2_0110_login/login,浏览器显示 login.jsp 页面,如图 10.2 所示。

3. 任务总结

通过本任务,实践了如何搭建 Struts 2 简单应用。

10.6　实践任务 2：配置 Action 实现请求与视图分离

1. 任务说明

本任务以论坛管理系统为例，体会 Struts 2 框架如何分离请求与视图。

修改实践任务一的工程，当用户输入请求 http://localhost:8080/struts2_0110_login/login，浏览器打开后台登录页面，如图 10.2 所示。

2. 任务实施

(1) 创建视图层 jsp 页面。在工程 struts2_0110_login 的 WebRoot 路径下，创建 main 文件夹，文件夹中创建 login.jsp 文件，编写主要代码如下：

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
...
<html>
  <head>
    ...
    <title>Login</title>
    ...
  </head>

  <body>
    这是登录页面。<br/>
  </body>
</html>
```

(2) 修改配置文件 struts.xml。修改<struts>和</struts>标签之间的文件路径代码，如下所示。

```
<package name="default" namespace="/" extends="struts-default">
    <action name="login">
        <result>
            /main/login.jsp
        </result>
    </action>
</package>
```

(3) 测试。部署工程，在浏览器中输入请求代码 http://localhost:8080/struts2_0110_login/login，浏览器显示 login.jsp 页面，如图 10.2 所示。

3. 任务总结

通过本任务，实践了 Struts 2 框架分离请求与视图，使程序更加灵活。

10.7　拓展任务

1. 使用 Struts 2 框架实现后台登录界面请求处理

任务建议：

根据 10.3 节的讲解，在 struts2_0110_login 工程的基础上，使用 Struts 2 框架实现论坛管理系统后台登录界面请求的处理。

2. 创建应用 Struts 2 框架的论坛管理系统工程

建议步骤：

(1) 创建 Web 应用——BBS_MS_0100。

(2) 搭建 Struts 2 框架开发环境。

(3) 创建 index.jsp 和 login.jsp 文件，编写文件代码。

(4) 编写 struts.xml 文件，配置处理打开 index.jsp 和 login.jsp 文件请求的 Action。

(5) 部署工程。

(6) 调试系统：打开浏览器，输入客户端请求，测试程序，如有错误，进行修改。

10.8 本章小结

本章通过手动和使用 IDE 工具两种方式，搭建一个最简单的 Struts 2 应用，以论坛管理系统的登录页面为例，演示了搭建最简单的 Struts 2 应用的方法和步骤，了解了 Struts 2 应用的架构，学习了 Struts 2 框架的核心控制器 StrutsPrepareAndExecuteFilter 及查看其源代码的方法，并理解了 Struts 2 框架如何将请求与视图分开，了解了 Struts 2 框架处理请求的简单流程。

另外，通过练习，对 struts.xml 文件配置也有所了解，了解了 package 元素中 namespace 属性、action 元素中 name 属性和 result 元素的用法和作用。

第11章

Action应用详解

本章学习Action的创建和使用。Struts框架的应用核心是Action,开发人员需要编辑大量的Action类,并在struts.xml文件中配置Action。Action类主要功能是接收请求、参数等信息,调用业务逻辑处理请求,返回逻辑视图。

本章要点:

- 开发模式的设置
- Action类的作用
- 实现Action类的三种方法
- ActionSupport类的组成和使用方法
- 在Action类中定义多个方法并调用
- 通配符的使用
- 默认Action的设置
- Action接收参数的三种方法

11.1 开发模式的设置

11.1.1 开发模式简介

开发模式的设置并不属于Action的内容,但是因为在学习Action的过程中,经常需要修改struts.xml文件,因此先学习这部分知识。

在开发系统过程中,经常需要修改配置文件,而每次修改配置文件,都需要重新启动Tomcat更新系统的部署,费时费力。

将环境设置为开发模式后,修改配置文件后不必重启Tomcat,只要保存配置文件即可更新。

开发模式的设置方法如下。

在struts.xml的<struts>和</struts>标签之间插入constant标签,设置name属性值为struts.devMode,value属性值为true,代码如下:

```
<constant name="struts.devMode" value="true" />
```

11.1.2 实践任务1:设置开发模式

1. 任务说明

本任务以论坛管理系统的后台用户管理模块为例,实践开发模式的设置。

先设置开发模式，并设置当用户提交请求 http://localhost:8080/struts2_0200_action/login，浏览器打开后台登录页面，如图 11.1 所示；修改配置文件代码并保存，不重新启动 Tomcat，实现自动更新，当修改用户提交请求为 http://localhost:8080/ struts2_0200_action/main/login，浏览器打开后台登录页面，如图 11.2 所示。



图 11.1 开发模式修改配置前



图 11.2 开发模式修改配置后

2. 任务实施

(1) 创建 Struts 应用。使用 MyEclipse 创建 Web 应用，名为 struts2_0200_action，配置 Tomcat 6.0 和 JDK 1.6，增加 Struts 支持。

(2) 创建视图层 jsp 页面。将 WebRoot 路径下的 index.jsp 文件改名为 login.jsp，主要代码如下：

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
...
<html>
  <head>
    ...
    <title>Login</title>
    ...
  </head>

  <body>
      这是后台登录界面
      <form action="" method="get">
          用户名：<input type="text"/>
          密码：<input type="password"/>
          <input type="submit" value="登录"/>
      </form>
  </body>
</html>
```

(3) 编写配置文件 struts.xml。打开配置文件 struts.xml，设置开发模式和配置 Action，代码如下：

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
    "http://struts.apache.org/dtds/struts-2.0.dtd">

<struts>
    <constant name="struts.devMode" value="true" />
    <package name="default" namespace="/" extends="struts-default">
        <action name="login">
            <result>
                /Login.jsp
            </result>
        </action>
    </package>
</struts>
```

(4) 启动 Tomcat，部署工程，测试。启动 Tomcat，部署工程，在浏览器的地址栏中输入请求 http://localhost:8080/struts2_0200_action/login，浏览器显示 login.jsp 页面，如图 11.1 所示。

(5) 修改配置文件 struts.xml。修改包的命名空间为/main，并保存 struts.xml 文件，代码如下：

```
<package name="default" namespace="/main" extends="struts-default">
    <action name="login">
        <result>
            /Login.jsp
        </result>
    </action>
</package>
```

(6) 测试。不重启 Tomcat，在浏览器中输入以下请求代码。

```
http://localhost:8080/struts2_0200_action/main/login
```

浏览器显示 Login.jsp 页面，如图 11.2 所示。

3. 任务总结

通过本任务，实践了设置开发模式，不必重启 Tomcat，修改配置文件 struts.xml 并保存后，Struts 应用自动更新。

11.2　Action 类的作用

Struts 框架的应用核心是 Action。开发人员在开发 Struts 应用的过程，就是编写大量的 Action 类，并在 struts.xml 文件中配置 Action 的过程。Action 类通过获取请求、参

数等信息，调用业务逻辑处理请求，返回逻辑视图，最终完成它的功能。

下面我们学习如何创建 Action。

11.3 实现 Action 类的三种方式

实现 Action 类共有三种方式：使用普通的 POJO 实现、通过实现 Action 接口实现、通过继承 ActionSupport 类实现。其中第三种最常用。

11.3.1 使用普通 POJO 实现 Action 的方法

Struts 2 框架采用低侵入式的设计，Action 类可以是一个普通的 POJO，可以不继承任何类或实现任何接口就可以实现，从而有很好的代码复用性，也便于代码的测试。

例如，下面代码就是一个 Action 类。

```
package com.BBS.struts2.action;
public class Login1Action {
    public String execute() {
        return "success";
    }
}
```

其中，execute()方法的方法名和返回值类型都不可以改变，这是有原因的，我们接下来会解释。

要调用 Login1Action 类的 execute 方法，在 struts. xml 文件的<package>标签和<package/>标签之间，编写以下配置。

```
<action name="login1" class="com.BBS.struts2.action.Login1Action">
   <result name="success">
        /index.jsp
   </result>
</action>
```

其中，action 标签的 class 属性值指定被调用的 Action 类，包括类的包名和类名，com. BBS. struts2. action 是包名，Login1Action 是 Action 类名。

当客户端提交 Action 对应的请求时，execute()方法会被自动调用。如果在定义 Action 类时，定义的方法名不是 execute，则无法自动调用，因此方法名必须为 execute。

execute()方法返回字符串"success"，就是 Action 类返回的逻辑视图名——每个 Action 类在处理完成之后，都会返回一个字符串类型的结果，这个结果就是一个逻辑视图名。因此，execute()方法的返回值类型必须是 String 类型。

在配置文件 struts. xml 中，<result>元素配置逻辑视图和物理视图的映射关系。以下配置代码配置名为 success 的逻辑视图与物理视图路径/index. jsp 之间的映射关系，

因此，浏览器显示/index.jsp 所指定的网页文件。

```
<result name="success">
        /index.jsp
</result>
```

另外，因为 name 属性值默认为 success，所以 result 元素的 name 属性设置也可以省略，<result>元素的配置代码可以如下编写。

```
<result>
    /index.jsp
</result>
```

11.3.2　实践任务 2：使用普通 POJO 实现 Action

1. 任务说明

本任务以论坛管理系统的后台用户管理模块为例，实践使用普通的 POJO 实现 Action。

当用户登录系统时，需要调用 Action 类处理用户的登录请求，调用业务逻辑组件判断用户是否合法，如果是合法用户，则进入系统主页。

当用户在登录界面，如图 11.2 所示，单击登录表单的"登录"按钮时，调用 Login1Action 类，处理请求，显示后台主页，如图 11.3 所示。

2. 任务实施

(1) 创建视图层 JSP 页面。打开上个实践任务创建的 Struts 应用"struts2_0200_action"，创建 index.jsp 文件，其主要代码如下：

```
<body>
  这是后台主页<br/>
</body>
```

(2) 创建 Action。在 src 路径下，创建包 com.BBS.struts2.action，在包下创建类 Login1Action，如图 11.4 所示。



图 11.3　后台主页



图 11.4　创建 Login1Action 类

Login1Action 类代码如下：

```
package com.BBS.struts2.action;
public class Login1Action {
    public String execute() {
        System.out.println("------调用 Login1Action 类------");
```

```
            return "success";
        }
    }
```

其中,"System. out. println("------调用 Login1Action 类------");"语句处,在实际开发中,应该是调用业务逻辑组件相应方法的代码,本任务主要实践如何创建和调用 Action 类,因此此处省略调用业务逻辑组件的代码,用输出语句代替。

(3) 在配置文件 struts. xml 中配置 Action。打开 struts. xml,配置 Action,代码如下:

```
<struts>
<package name="default" namespace="/main" extends="struts-default">
...
     <action name="login1" class="com.BBS.struts2.action.Login1Action">
       <result name="success">
            /index.jsp
       </result>
     </action>
</package>
</struts>
```

(4) 修改 JSP 页,提交请求。打开 Login. jsp 页,设置表单的 action 属性值为"main/login1",代码如下:

```
<form action="main/login1"   method="get">
    用户名: <input type="text"/>
    密码: <input type="password"/>
    <input type="submit" value="登录"/>
</form>
```

(5) 测试。部署工程,在浏览器中输入请求代码 http://localhost:8080/struts2_0200_action/main/login,浏览器显示 Login. jsp 页面,如图 11. 2 所示,单击"登录"按钮,提交表单,浏览器显示 index. jsp 页面,控制台输出字符串"------调用 Login1Action 类------",表示调用了 Login1Action 类的 execute()方法。

3. 任务总结

通过本任务,实践了使用普通 POJO 实现 Action 的方法,并实践了如何调用 Action 类的 execute()方法。

11.3.3 通过实现 Action 接口实现

1. 创建 Action

Struts 框架的 Action 类也可以通过实现 Action 接口来实现。

例如,下面代码就是一个 Action 类。

```
package com.BBS.struts2.action;
import com.opensymphony.xwork2.Action;
public class Login2Action implements Action{
```

```
        public String execute() {
            return    SUCCESS;
        }
}
```

2. Action 接口

Action 接口在 xwork-core-2.2.1.jar 的 com.opensymphony.xwork2 包下，可以在工程中打开，如图 11.5 所示。

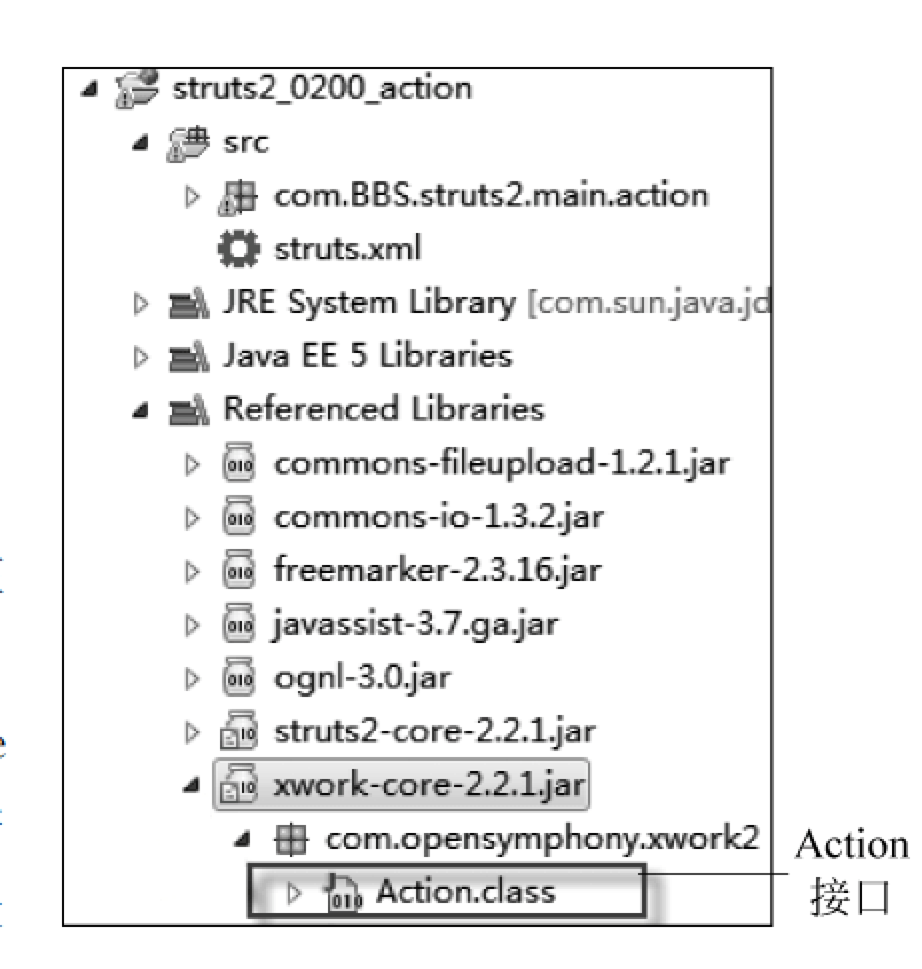


图 11.5　Action 接口

可以通过以下步骤查看 Action 接口的源代码。

(1) 在工程中右键单击 xwork-core-2.2.1.jar，在弹出的菜单中选择 Properties 命令，如图 11.6 所示。

(2) 在弹出的对话框中，选择 Java Source Attachment 选项，单击右侧的 External Folder 按钮，选择 XWork 的源码，我放在了 D 盘的根目录中，则路径为 D:/xwork-2.0.7，如图 11.7 所示。

(3) 单击 OK 按钮，确定设置。

(4) 在工程中，打开 xwork-core-2.2.1.jar，在 com.opensymphony.xwork2 包中双击 Action.class，就可以看到其源代码，如图 11.8 所示。

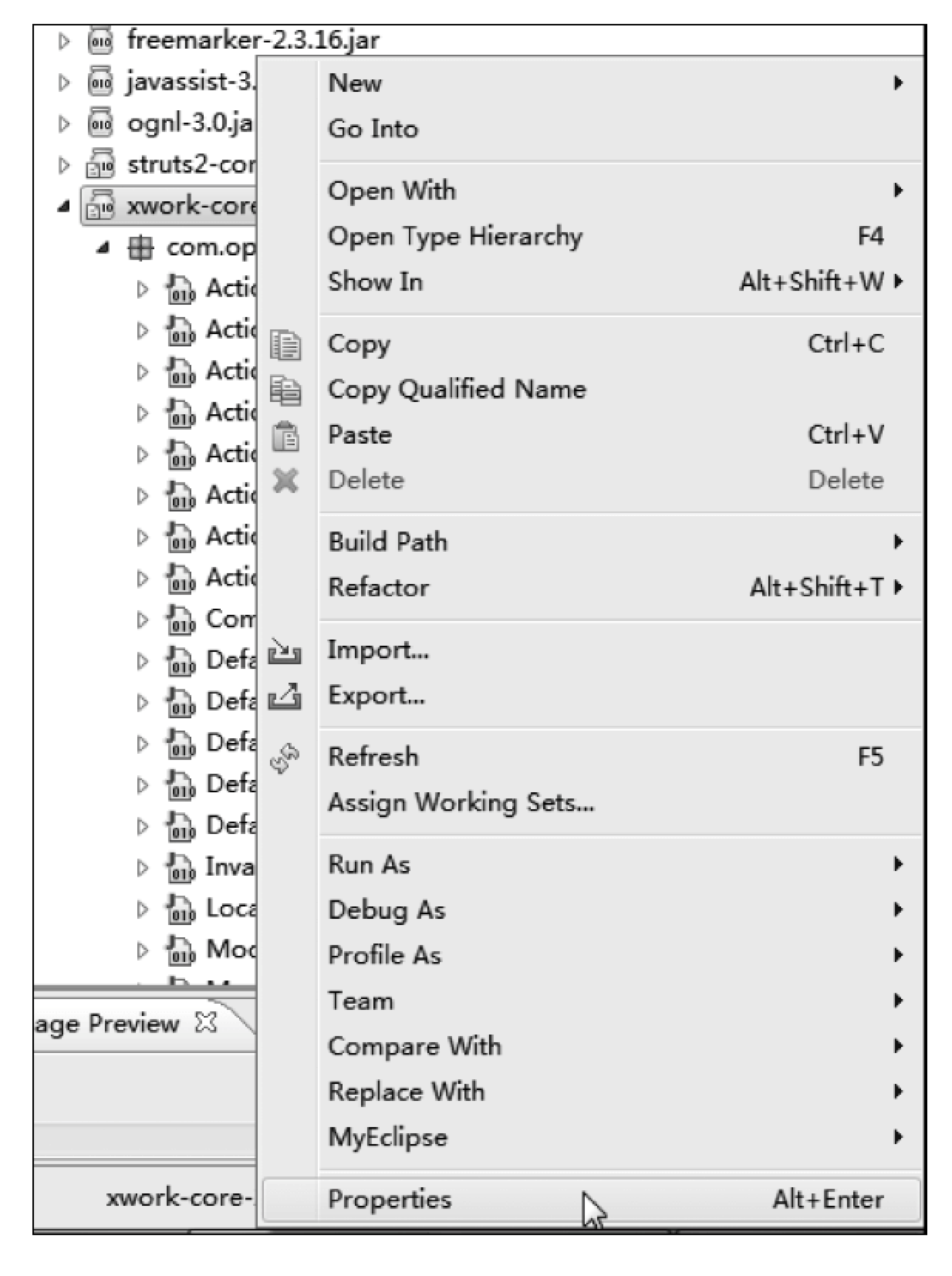


图 11.6　选择 Properties 命令

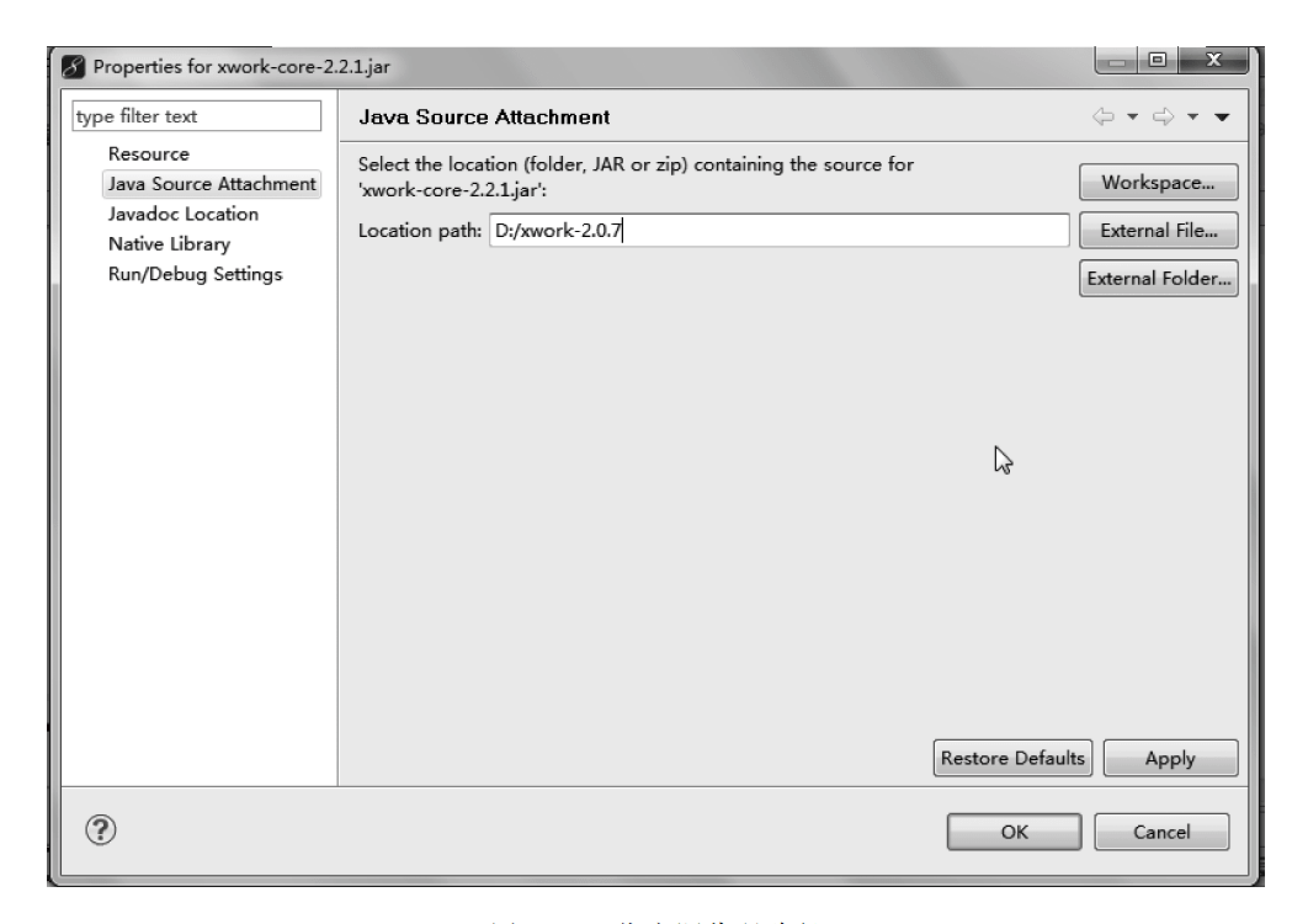


图 11.7　指定源代码路径

其主要源代码如下：

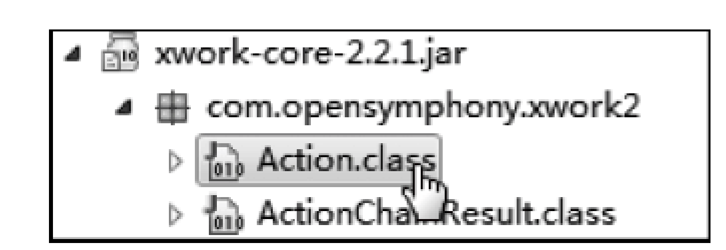


图 11.8　双击 Action. class

```
public interface Action {
    public static final String SUCCESS = "success";
    public static final String NONE = "none";
    public static final String ERROR = "error";
    public static final String INPUT = "input";
    public static final String LOGIN = "login";
    public String execute() throws Exception;
}
```

Action 接口定义了 5 个字符串类型的符号常量：SUCCESS、NONE、ERROR、INPUT、LOGIN，它们统一 execute()方法的返回值，使 execute()方法的返回值标准化。例如处理成功，返回符号常量 SUCCESS，而避免返回字符串常量 success，因为在程序中应该尽量返回符号常量。

Action 接口还定义了抽象方法 execute()，规定了 Action 类应该包含一个 execute()方法，该方法返回一个字符串。

因此，在上面定义的 Login2Acion 的代码中，实现了 execute()方法，其返回值为符号常量 SUCCESS，其值是字符串"success"。

3. 配置 Action

要调用 Login2Action 类的 execute()方法，在 struts. xml 文件的<package>标签和<package/>标签之间，编写以下配置。

```
<action name="login2" class="com.BBS.struts2.action.Login2Action">
    <result name="success">
        /index.jsp
    </result>
</action>
```

其中,<result>标签的 name 属性值是"success",即 Action 返回的符号常量 SUCCESS 的值,对应的物理视图为 index.jsp。因此,浏览器显示/index.jsp 所指定的网页文件。

11.3.4　实践任务 3:通过实现 Action 接口实现 Action

1. 任务说明

本任务以论坛管理系统的后台用户管理模块为例,实践通过实现 Action 接口实现 Action 类。

当用户登录系统时,需要调用 Action 类处理用户的登录请求,调用业务逻辑组件判断用户是否合法,如果是合法用户,则进入系统主页。

当用户在登录界面,如图 11.2 所示,单击"登录"按钮时,调用 Login2Action 类,处理请求,显示后台主页,如图 11.3 所示。

2. 任务实施

(1) 创建 Action。打开上个实践任务创建的 Struts 应用"struts2_0200_action",在 com.BBS.struts2.action 包下创建类 Login2Action。

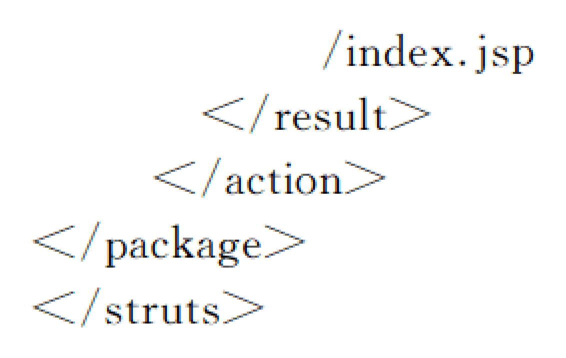
Login2Action 类代码如下:

```
package com.BBS.struts2.action;
import com.opensymphony.xwork2.Action;
public class Login2Action implements Action{
    public String execute() {
        System.out.println("------调用 Login2Action 类------");
        return   SUCCESS;
    }
}
```

其中,"System.out.println("------调用 Login2Action 类------");"语句处,在实际开发中,应该是调用业务逻辑组件的代码,本任务主要实践如何创建和调用 Action 类,因此此处省略调用业务逻辑组件的代码,用输出语句代替。

(2) 在配置文件 struts.xml 在配置 Action。打开 struts.xml,配置 Action,代码如下:

```
<struts>
<package name="default" namespace="/main" extends="struts-default">
...
     <action name="login2" class="com.BBS.struts2.action.Login2Action">
       <result name="success">
```



```
                /index.jsp
            </result>
        </action>
    </package>
</struts>
```

（3）修改 JSP 页，提交请求。打开 Login.jsp 页，修改表单的 action 属性值为"main/login2"。

（4）测试。部署工程，在浏览器中输入请求代码 http://localhost:8080/struts2_0200_action/main/login，浏览器显示 Login.jsp 页面，如图 11.2 所示，单击“登录”按钮，提交表单，浏览器显示 index.jsp 页面，控制台输出字符串“------调用 Login2Action 类------”，表示调用了 Login2Action 类的 execute()方法。

3. 任务总结

通过本任务，实践了通过实现 Action 接口实现 Action 的方法。

11.3.5 通过继承 ActionSupport 类实现

1. 创建 Action

Struts 框架的 Action 类还可以通过继承 ActionSupport 类来实现。

例如，下面代码就是一个 Action 类。

```
package com.BBS.struts2.action;
import com.opensymphony.xwork2.ActionSupport;
public class Login3Action extends ActionSupport{
    public  String execute(){
        return SUCCESS;
    }
}
```

2. ActionSupport 类

经过 11.2.2 小节第 2 条中设置源代码路径，可以直接查看所有 xwork-core-2.2.1.jar 中的类的源代码。在工程中，打开 xwork-core-2.2.1.jar，在 com.opensymphony.xwork2 包中双击 ActionSupport.class，就可以看到其源代码。

因为 ActionSupport 类代码较多，下面仅列出其主要代码片段。

```
public class ActionSupport implements Action, Validateable, ValidationAware, TextProvider,
LocaleProvider, Serializable {
    //收集校验错误码的方法
    public void setActionErrors(Collection errorMessages) {
        validationAware.setActionErrors(errorMessages);
    }
    //返回校验错误的方法
    public Collection getActionErrors() {
        return validationAware.getActionErrors();
    }
```

```
public void setActionMessages(Collection messages) {
    validationAware.setActionMessages(messages);
}

public Collection getActionMessages() {
    return validationAware.getActionMessages();
}
//设置表单域校验错误的信息
public void setFieldErrors(Map errorMap) {
    validationAware.setFieldErrors(errorMap);
}
//返回表单域校验错误的信息
public Map getFieldErrors() {
    return validationAware.getFieldErrors();
}
//添加关于错误的信息
public void addActionError(String anErrorMessage) {
    validationAware.addActionError(anErrorMessage);
}

public void addActionMessage(String aMessage) {
    validationAware.addActionMessage(aMessage);
}
//添加字段校验失败的出错信息
public void addFieldError(String fieldName, String errorMessage) {
    validationAware.addFieldError(fieldName, errorMessage);
}
...
//默认的 input 方法,直接返回字符串 input
public String input() throws Exception {
    return INPUT;
}

public String doDefault() throws Exception {
    return SUCCESS;
}
...
//默认的处理用户请求的方法,直接返回字符串 success
public String execute() throws Exception {
    return SUCCESS;
}
...
//清除所有关于错误的信息
public void clearErrorsAndMessages() {
    validationAware.clearErrorsAndMessages();
}
...
//空的数据输入校验方法
```

```
        public void validate() {
        }
        ...
    }
```

ActionSupport 类实现了 Action 接口，也就继承了 Action 接口的 5 个符号常量，而且 ActionSupport 类实现了 execute()方法，返回符号常量 SUCCESS。

此外，ActionSupport 类还实现了 Validateable、ValidationAware 等接口，创建了很多默认方法，例如数据输入校验、错误信息处理等方法。

ActionSupport 类完全可以作为 Struts 应用的 Action 处理类，当 struts.xml 文件中配置代码并未指定 Action 处理类时，例如：

```
<action name="login" >
    <result>
        /Login.jsp
    </result>
</action>
```

Struts 2 框架会自动创建 ActionSupport 类的代理，调用 ActionSupport 类的 execute()方法处理请求，返回符号常量 SUCCESS，<result>标签的 name 属性默认值为字符串 success，因此上面配置代码结果会在浏览器中显示 Login.jsp 页面。

创建 Action 处理类时，继承 ActionSupport 类，也就继承了 execute 等多个方法，会大大简化 Action 的开发。因此，通过继承 ActionSupport 类实现 Action 的方式是在实际开发中最常用的方式。

3. 配置 Action

要调用 Login3Action 类的 execute()方法，在 struts.xml 文件的<package>标签和<package/>标签之间，编写以下配置代码。

```
<action name="login3" class="com.BBS.struts2.action.Login3Action">
    <result>
        /index.jsp
    </result>
</action>
```

其中，<result>标签的 name 属性值是"success"，即 Action 返回的符号常量 SUCCESS 的值，对应的物理视图为 index.jsp。因此，浏览器显示"/index.jsp"所指定的网页文件。

11.3.6 实践任务 4：通过继承 ActionSupport 类实现 Action

1. 任务说明

本任务以论坛管理系统的后台用户管理模块为例，实践通过继承 ActionSupport 类实现 Action。

当用户在登录界面(见图 11.2)单击"登录"按钮时，调用 Login3Action 类，处理请求，显示后台主页，如图 11.3 所示。

2. 任务实施

(1) 创建 Action。打开上个实践任务创建的 Struts 应用 struts2_0200_action，在 com. BBS. struts2. action 包下创建类 Login3Action。

Login3Action 类代码如下：

```
package com.BBS.struts2.action;
import com.opensymphony.xwork2.ActionSupport;
public class Login3Action extends ActionSupport{
    public  String execute(){
        System.out.println("------调用 Login3Action 类------");
        return SUCCESS;
    }
}
```

其中，“System. out. println("------调用 Login3Action 类------");”语句处，在实际开发中，应该是调用业务逻辑组件的代码，本任务主要实践如何创建和调用 Action 类，因此此处省略调用业务逻辑组件的代码，用输出语句代替。

(2) 在配置文件 struts. xml 在配置 Action。打开 struts. xml，配置 Action，代码如下：

```
<struts>
<package name="default" namespace="/main" extends="struts-default">
...
    <action name="login3" class="com.BBS.struts2.action.Login3Action">
      <result>
          /index.jsp
      </result>
    </action>
</package>
</struts>
```

(3) 修改 JSP 页，提交请求。打开 login. jsp 页，修改表单的 action 属性值为"main/login3"。

(4) 测试。部署工程，在浏览器中输入请求代码 http://localhost:8080/struts2_0200_action/main/login，浏览器显示 login. jsp 页面，如图 11. 2 所示，单击“登录”按钮，提交表单，浏览器显示 index. jsp 页面，控制台输出字符串“------调用 Login3Action 类------”，表示调用了 Login3Action 类的 execute()方法。

3. 任务总结

通过本任务，实践了通过继承 ActionSupport 类实现 Action 的方法。

11.4 调用 Action 类中的指定方法

一个 Action 类中通常不会只有一个 execute()方法。Action 类需要编写多个方法，用于处理客户端的不同请求。例如，用户管理模块中的增加用户、删除用户、修改用户和

查询用户需要四个方法来实现。因此,一个 Action 中一般需要定义多个方法。那么,如何根据不同的请求调用指定的方法呢?

调用指定方法有两种方式:静态调用和动态调用。

11.4.1 在 Action 类创建多个方法

在 Action 类中不仅可以创建 execute()方法,还可以创建多个其他名称的方法。用于处理的请求方法,方法名可以任意,其参数的类型和个数也没有要求,但是其返回值类型必须为字符串类型,因为 Action 中处理请求的方法要求返回一个字符串作为逻辑视图名。

例如,定义处理用户管理模块请求的 Action 类,包括处理添加用户、删除用户、修改用户信息、显示用户信息列表请求的方法,代码如下:

```
import com.opensymphony.xwork2.ActionSupport;
public class UserAction extends ActionSupport{
    //处理添加用户请求的方法
    public String add() {
        return SUCCESS;
    }
    //处理删除用户请求的方法
    public String delete() {
        return SUCCESS;
    }
    //处理修改用户请求的方法
    public String modify() {
        return SUCCESS;
    }
    //处理显示用户列表请求的方法
    public String list() {
        return SUCCESS;
    }
}
```

11.4.2 静态调用 Action 类中的指定方法

通过在配置文件中定义多个逻辑 Action,并为<action>元素指定 method 属性,属性名为 Action 类中的方法名,可以调用指定的方法。

例如,调用 UserAction 的指定方法,其配置代码如下:

```
<package name="default" namespace="/main" extends="struts-default">
<!--调用 UserAction 中的 add 方法-->
<action name="User_add" class="com.BBS.struts2.action.UserAction" method="add">
   <result>
        /User_add_success.jsp
   </result>
```

```
</action>
<!--调用 UserAction 中的 list 方法-->
<action name="User_list" class="com.BBS.struts2.action.UserAction" method="list">
    <result>
        /User_list_success.jsp
    </result>
</action>
<!--调用 UserAction 中的 delete 方法-->
<action name="User_delete" class="com.BBS.struts2.action.UserAction"
    method="delete">
    <result>
        /User_delete_success.jsp
    </result>
</action>
<!--调用 UserAction 中的 modify 方法-->
<action name="User_modify" class="com.BBS.struts2.action.UserAction"
    method="modify">
    <result>
        /User_modify_success.jsp
    </result>
</action>
</package>
```

<! --和-->之间的是注释语句。

调用各方法的请求代码如下。

(1) 调用 UserAction 类 add 方法的请求为“工程路径/main/User_add”。

(2) 调用 UserAction 类 list 方法的请求为“工程路径/main/User_list”。

(3) 调用 UserAction 类 delete 方法的请求为“工程路径/main/User_delete”。

(4) 调用 UserAction 类 modify 方法的请求为“工程路径/main/User_modify”。

11.4.3　实践任务 5：静态调用 Action 中的指定方法

1. 任务说明

本任务以论坛管理系统的后台用户管理模块为例，实践使用静态方式调用 Action 中的指定方法。

当用户在后台主页(见图 11.9)单击“显示用户列表”的超链接时，打开 User_list_success.jsp 页面，显示用户列表，如图 11.10 所示。

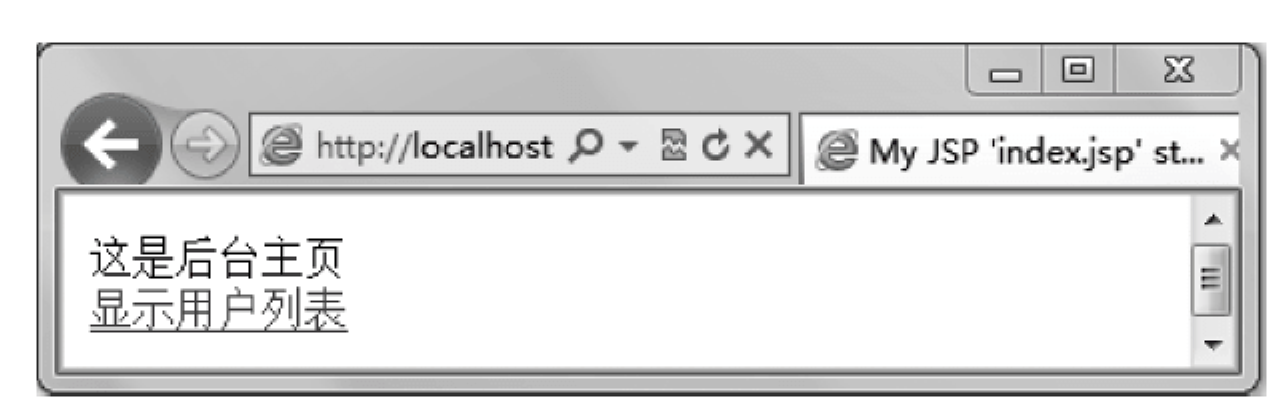


图 11.9　后台主页



图 11.10　用户列表管理页

当用户在添加用户页面单击“添加”按钮(见图 11.11),就会打开 User_add_success.jsp 页面,显示用户添加成功,如图 11.12 所示。

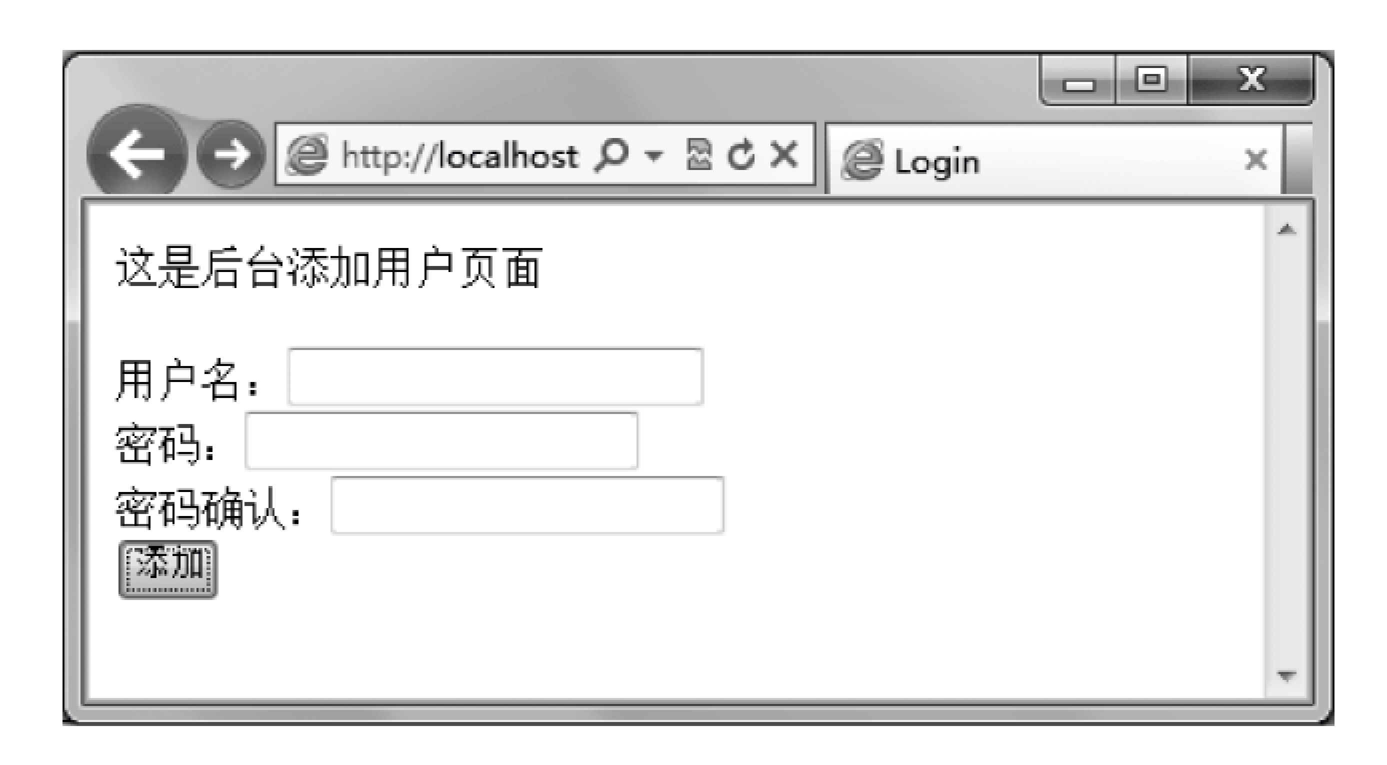


图 11.11　用户添加输入页

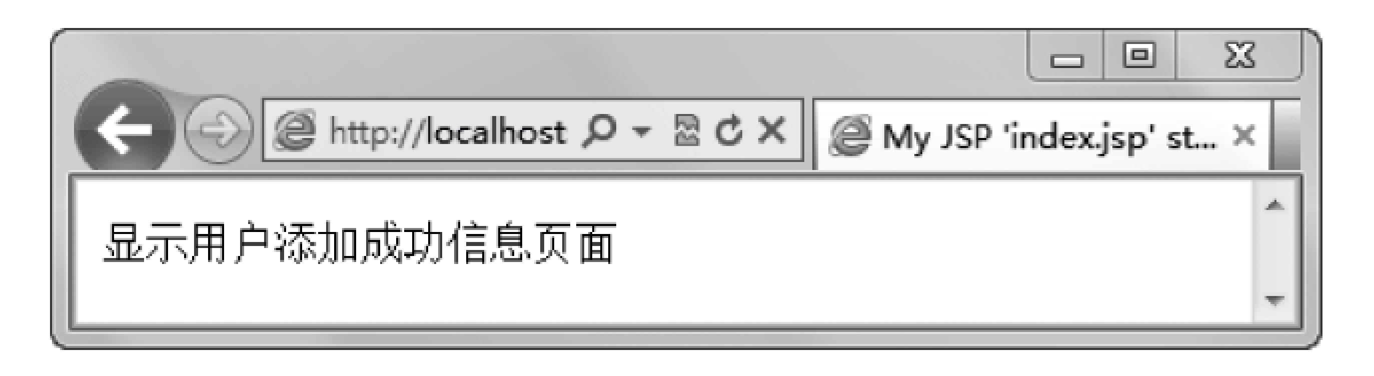


图 11.12　用户添加成功页

2. 任务实施

(1) 创建视图层文件。打开工程 struts2_0200_action,创建视图层文件 User_list_success.jsp、User_add_input.jsp 和 User_add_success.jsp。

① 创建 User_list_success.jsp,输入以下代码。

```
<body>
    用户列表管理页面 <br>
    用户 1 信息<br>
    用户 2 信息<br>
    用户 3 信息<br>
    用户 4 信息 <br>
    用户 5 信息 <br>
```

```
</body>
```

用户列表管理页面应该从数据库中读取用户信息并显示列表，因为本任务并没有编写访问数据库的相关代码，因此使用以下代码模拟显示用户信息列表。

```
用户 1 信息<br>
用户 2 信息<br>
用户 3 信息<br>
用户 4 信息 <br>
用户 5 信息 <br>
```

② 创建 User_add_input. jsp，输入以下代码。

```
<body>
  这是后台添加用户页面
  <form action="" method="get">
  用户名：<input type="text"/><br/>
  密码：<input type="password"/><br/>
  密码确认：<input type="password"/><br/>
  <input type="submit" value="添加"/>
  </form>
</body>
```

③ 创建 User_add_success. jsp，输入以下代码。

```
<body>
    显示用户添加成功信息页面 <br>
</body>
```

(2) 创建 Action。打开上个实践任务创建的 Struts 应用 struts2_0200_action，在 com. BBS. struts2. action 包下创建类 UserAction。

UserAction 类的代码如下：

```
package com.BBS.struts2.action;
import com.opensymphony.xwork2.ActionSupport;
public class UserAction extends ActionSupport{
//处理添加用户请求的方法
    public String add() {
        System.out.println("------调用 UserAction 类的 add 方法------");
        return SUCCESS;
    }
    //处理删除用户请求的方法
    public String delete() {
        System.out.println("------调用 UserAction 类的 delete 方法------");
        return SUCCESS;
    }
    //处理修改用户请求的方法
    public String modify() {
        System.out.println("------调用 UserAction 类的 modify 方法------");
        return SUCCESS;
```

```
    }
    //处理显示用户列表请求的方法
    public String list() {
        System.out.println("------调用 UserAction 类的 list 方法------");
        return SUCCESS;
    }
}
```

在实际开发中,“System.out.println();”语句处应该是调用业务逻辑组件的代码,本任务主要实践如何创建和调用 Action 类,因此此处省略调用业务逻辑组件的代码,用输出语句代替。

(3) 在配置文件 struts.xml 在配置 Action。打开 struts.xml,配置 Action,代码如下:

```
<struts>
<package name="default" namespace="/main" extends="struts-default">
...
<!--调用 UserAction 中的 add 方法-->
<action name="User_add" class="com.BBS.struts2.action.UserAction" method="add">
    <result>
        /User_add_success.jsp
    </result>
</action>
<!--调用 UserAction 中的 list 方法-->
<action name="User_list" class="com.BBS.struts2.action.UserAction" method="list">
    <result>
      /User_list_success.jsp
    </result>
</action>
</struts>
```

(4) 修改 JSP 页,提交请求。

① 打开 index.jsp 页,输入请求以下代码。

```
<body>
    这是后台主页<br>
<a href="main/User_list">显示用户列表</a>
</body>
```

② 打开 User_add_input.jsp,将表单的 action 属性值修改为"main/User_add"。

(5) 测试。部署工程,在浏览器中输入请求代码 http://localhost:8080/struts2_0200_action/main/login1,浏览器显示 index.jsp 页面,如图 11.9 所示,单击“显示用户列表”超链接,浏览器显示 User_list_success.jsp 页面,显示用户列表,如图 11.10 所示。控制台输出字符串“------调用 UserAction 类的 list 方法------”,表示调用了 UserAction 类的 list 方法。

在浏览器中输入请求代码 http://localhost:8080/struts2_0200_action/User_add_input.jsp,浏览器显示 User_add_input.jsp 页面,如图 11.11 所示,单击“添加”按钮,提交表

单，浏览器显示 User_add_success.jsp 页面，如图 11.12 所示。控制台输出字符串“------调用 UserAction 类的 add 方法------”，表示调用了 UserAction 类的 add 方法。

3. 任务总结

通过本任务，实践了在 Action 中定义多个方法，并使用静态方式调用指定的方法。

11.4.4　拓展任务

1. 使用 Struts 2 框架实现后台用户管理模块请求处理

建议步骤：

（1）打开 Struts 应用——struts2_0200_action。

（2）创建 User_delete_success.jsp、User_modify_success.jsp 和 User_modify_input.jsp 文件，编写文件代码。

（3）编写 struts.xml 文件，设置调用 delete 方法处理 User_delete 请求，显示 User_delete_success.jsp 页面，设置调用 modify 方法处理 User_modify 请求，显示 User_modify_success.jsp 页面。

（4）打开 User_list_success.jsp 文件编写删除的请求，打开 User_modify_input.jsp 文件，编写修改用户信息的请求。

（5）部署工程。

（6）调试系统。打开浏览器，输入客户端请求，测试程序，如有错误，进行修改。

2. 使用 Struts 2 框架实现后台主题管理模块请求处理

建议步骤：

（1）打开 Struts 应用——struts2_0200_action。

（2）创建 Theme_delete_success.jsp、Theme_add_success.jsp、Theme_add_input.jsp、Theme_list_success.jsp、Theme_modify_success.jsp 和 Theme_modify_input.jsp 文件，编写文件代码。

（3）创建 ThemeAction，编写添加、删除、修改、显示列表主题的方法。

（4）编写 struts.xml 文件，设置 Action，调用 ThemeAction 的指定方法。

（5）打开 index.jsp 文件编写显示主题列表的请求，打开 Theme_add_input.jsp 文件编写添加主题的请求，打开 Theme_list_success.jsp 文件编写删除主题的请求，打开 Theme_modify_input.jsp 文件，编写修改主题信息的请求。

（6）部署工程。

（7）调试系统。打开浏览器，输入客户端请求，测试程序，如有错误，进行修改。

11.4.5　动态调用 Action 类中的指定方法

静态方式调用时，需要创建多个逻辑 Action，使配置文件中的代码变得臃肿。

采用动态方式调用，可以减少配置文件中的代码量。

1. 采用动态方式调用

需要修改 Action 的代码，使各方法返回值不同，即 Action 类调用不同的方法，返回的逻辑视图不同。代码如下：

```
import com.opensymphony.xwork2.ActionSupport;
public class UserAction extends ActionSupport{
    //处理添加用户请求的方法
    public String add() {
        return "add";
    }
    //处理删除用户请求的方法
    public String delete() {
        return "delete";
    }
    //处理修改用户请求的方法
    public String modify() {
        return "modify";
    }
    //处理显示用户列表请求的方法
    public String list() {
        return "list";
    }
}
```

2. 根据不同的逻辑视图，配置其物理视图

调用 UserAction 的指定方法的配置代码如下：

```
<package name="default" namespace="/main" extends="struts-default">
<action name="User" class="com.BBS.struts2.action.UserAction">
   <result name="add">
        /User_add_success.jsp
   </result>
   <result name="delete">
        /User_delete_success.jsp
   </result>
   <result name="modify">
        /User_modify_success.jsp
   </result>
   <result name="list">
        /User_list_success.jsp
   </result>
</action>
</package>
```

3. 动态方式调用的格式

动态方式调用的格式如下：

```
Action 名!方法名
```

调用各方法的请求代码如下。

(1) 调用 UserAction 类 add 方法的请求为“工程路径/main/User！add”。

(2) 调用 UserAction 类 list 方法的请求为“工程路径/main/User！list”。

(3) 调用 UserAction 类 delete 方法的请求为“工程路径/main/User！delete”。

(4) 调用 UserAction 类 modify 方法的请求为“工程路径/main/User！modify”。

11.4.6　实践任务 6：动态方式调用 Action 中的指定方法

1. 任务说明

本任务以论坛管理系统的后台用户管理模块为例，实践使用动态方式调用 Action 中的指定方法。

当用户在后台主页（见图 11.9）单击“显示用户列表”的超链接时，打开 User_list_success.jsp 页面，显示用户列表。

当用户在添加用户页面（见图 11.11），单击“添加”按钮时，打开 User_add_success.jsp 页面，显示用户添加成功，如图 11.12 所示。

2. 任务实施

(1) 创建工程 BBS_Struts2_0300_ActionMethodTredns。

打开工程 struts2_0200_action，复制并粘贴工程，将其重命名为 BBS_Struts2_0300_ActionMethodTredns。

注意：别忘了修改其 Web Context-root 为/BBS_Struts2_0300_ActionMethodTredns。

修改的方法是在工程处右击，选择快捷菜单中的 Properties 命令，打开工程属性对话框。选择 MyEclipse 下的 Web 选项，修改其 Web Context-root 为/BBS_Struts2_0300_ActionMethodTredns，如图 11.13 所示。

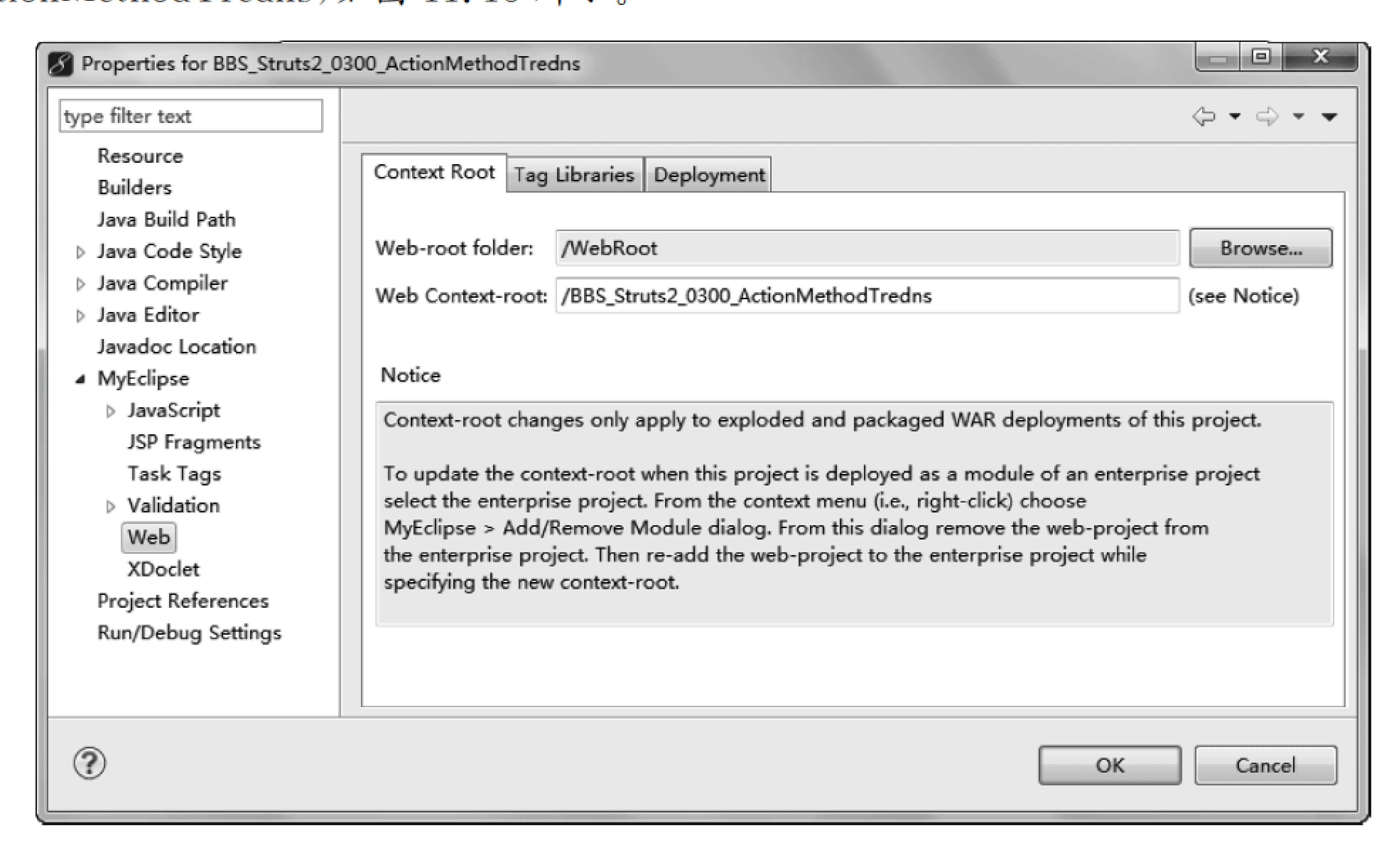


图 11.13　设置 Web Context-root

(2) 修改 Action。修改 UserAction 类代码,如下所示。

```
package com.BBS.struts2.action;
import com.opensymphony.xwork2.ActionSupport;
public class UserAction extends ActionSupport{
//处理添加用户请求的方法
    public String add() {
        System.out.println("------调用 UserAction 类的 add 方法------");
        return "add";
    }
    //处理删除用户请求的方法
    public String delete() {
        System.out.println("------调用 UserAction 类的 delete 方法------");
        return "delete";
    }
    //处理修改用户请求的方法
    public String modify() {
        System.out.println("------调用 UserAction 类的 modify 方法------");
        return "modify";
    }
    //处理显示用户列表请求的方法
    public String list() {
        System.out.println("------调用 UserAction 类的 list 方法------");
        return "list";
    }
}
```

在实际开发中,“System.out.println();”语句处应该是调用业务逻辑组件的代码,本任务主要实践如何创建和调用 Action 类,因此此处省略调用业务逻辑组件的代码,用输出语句代替。

(3) 在配置文件 struts.xml 在配置 Action。打开 struts.xml,配置 Action,代码如下:

```
<struts>
<package name="default" namespace="/main" extends="struts-default">
...
  <package name="default" namespace="/main" extends="struts-default">
<action name="User" class="com.BBS.struts2.action.UserAction">
   <result name="add">
        /User_add_success.jsp
   </result>
   <result name="list">
        /User_list_success.jsp
   </result>
</action>
</package>
</struts>
```

（4）修改 JSP 页，提交请求。

① 打开 index.jsp 页，输入如下请求代码。

```
<body>
  这是后台主页<br>
<a href="main/User!list">显示用户列表</a>
</body>
```

② 打开 User_add_input.jsp，将表单的 action 属性值修改为"main/User! add"。

（5）测试。部署工程，在浏览器中输入请求代码 http://localhost:8080/struts2_0200_action/main/login1，浏览器显示 index.jsp 页面，如图 11.9 所示，单击“显示用户列表”超链接，浏览器显示 User_list_success.jsp 页面，显示用户列表，如图 11.10 所示。控制台输出字符串“------调用 UserAction 类的 list 方法------”，表示调用了 UserAction 类的 list 方法。

在浏览器中输入请求代码 http://localhost:8080/struts2_0200_action/User_add_input.jsp，浏览器显示 User_add_input.jsp 页面，如图 11.11 所示，单击“添加”按钮，提交表单，浏览器显示 User_add_success.jsp 页面，如图 11.12 所示。控制台输出字符串“------调用 UserAction 类的 add 方法------”，表示调用了 UserAction 类的 add 方法。

3. 任务总结

通过本任务，实践了在 Action 中定义多个方法，并使用动态方式调用指定方法。

11.4.7　拓展任务

1. 使用 Struts 2 框架实现后台用户管理模块请求处理

建议步骤：

（1）打开工程——BBS_Struts2_0300_ActionMethodTrends。

（2）编写 struts.xml 文件，设置逻辑视图 delete 对应物理视图为 User_delete_success.jsp 页面，设置逻辑视图 modify 对应物理视图为 User_modify_success.jsp 页面。

（3）打开 User_list_success.jsp 文件编写删除的请求，打开 User_modify_input.jsp 文件，编写修改用户信息的请求。

（4）部署工程。

（5）调试系统。打开浏览器，输入客户端请求，测试程序，如有错误，进行修改。

2. 使用 Struts 2 框架实现后台主题管理模块请求处理

建议步骤：

（1）打开 Struts 应用——BBS_Struts2_0300_ActionMethodTrends。

（2）修改 ThemeAction，编写添加、删除、修改、显示列表主题的方法，使其返回逻辑视图不同。

（3）编写 struts.xml 文件，设置 Action。

(4) 打开 index. jsp 文件编写显示主题列表的请求，打开 Theme_add_input. jsp 文件编写添加主题的请求，打开 Theme_list_success. jsp 文件编写删除主题的请求，打开 Theme_modify_input. jsp 文件，编写修改主题信息的请求。

(5) 部署工程。

(6) 调试系统。打开浏览器，输入客户端请求，测试程序，如有错误，进行修改。

11.4.8 实训 1：初步搭建论坛管理系统后台用户管理模块框架

1. 实训要求

根据界面跳转图(见图 11.14)，完成后台用户管理模块中的请求提交、Action 接收并处理请求和显示处理结果页，包括后台用户登录、添加用户、删除用户、修改用户和显示用户列表。

2. 建议步骤

(1) 创建 Struts 应用。

(2) 创建视图层文件 Login. jsp、index. jsp、User_list_success. jsp、User_delete_success. jsp、User_add. jsp、User_add_success. jsp、User_modify. jsp、User_modify_success. jsp、default. jsp 等，并编写页面提示信息代码。

(3) 实现控制层 Action 类——UserAction，方法包括 login、list、delete、add、modify、modifyInput 等。

(4) 编写配置文件 struts. xml，设置开发模式，配置 namespace、action、result。

(5) 编写视图层文件代码，编写提交请求的代码。

(6) 调试系统：部署工程，测试程序，如有错误，进行修改。

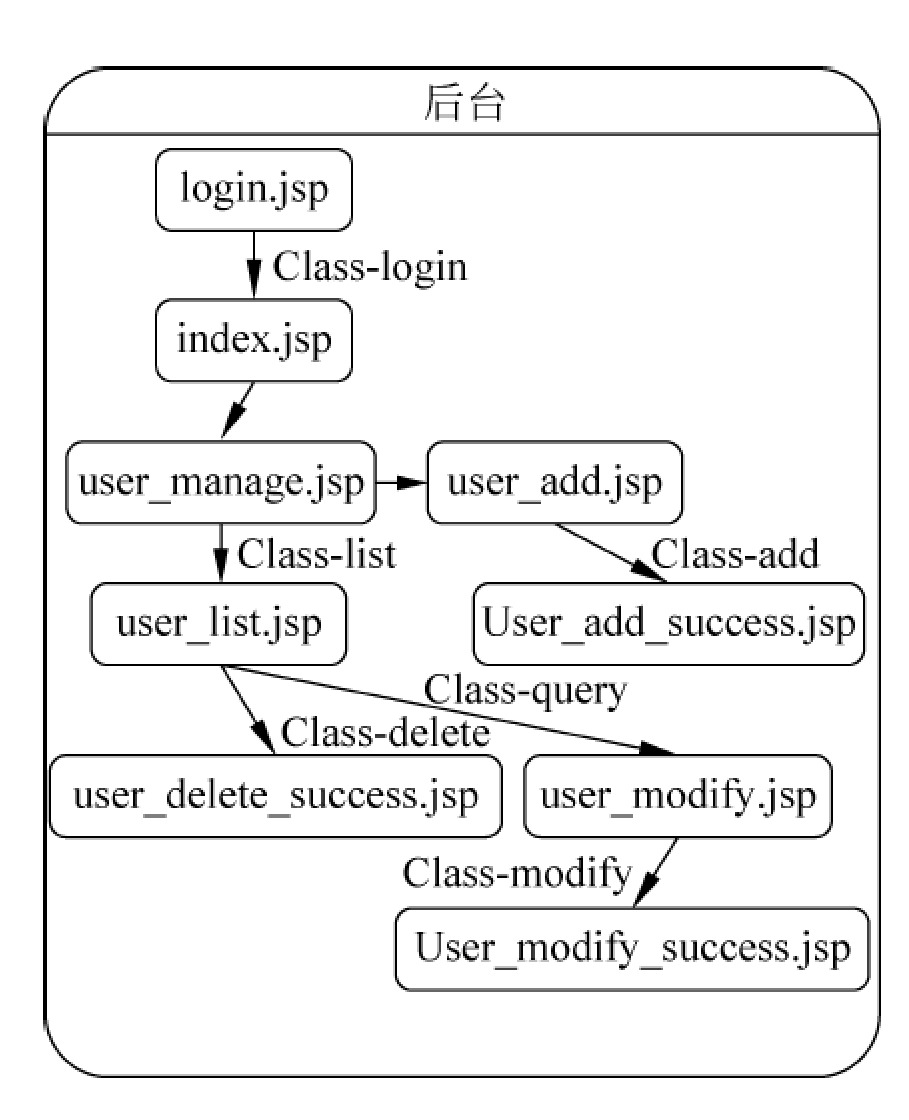


图 11.14 界面跳转图

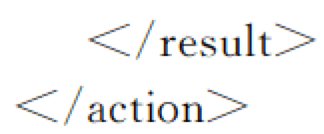
11.5 使用通配符

11.5.1 通配符简介

在配置<action>元素时，需要指定 name、class 和 method 属性，其中 name 属性可以支持通配符 *，然后在 class 和 method 属性使用表达式，以对应 name 属性中的通配符。使用通配符是另一个动态方法调用，当我们使用通配符定义<action>元素的 name 属性时，相当于定义了多个逻辑 Action，可以大量减少配置代码。

例如，有如下配置代码。

```
<!--调用 UserAction 中的 add 方法-->
<action name="User_add" class="com.BBS.struts2.action.UserAction" method="add">
    <result>
        /User_add_success.jsp
    </result>
</action>
<!--调用 UserAction 中的 list 方法-->
<action name="User_list" class="com.BBS.struts2.action.UserAction" method="list">
    <result>
        /User_list_success.jsp
    </result>
</action>
<!--调用 UserAction 中的 delete 方法-->
<action name="User_delete" class="com.BBS.struts2.action.UserAction" method="delete">
    <result>
        /User_delete_success.jsp
    </result>
</action>
<!--调用 UserAction 中的 modify 方法-->
<action name="User_modify" class="com.BBS.struts2.action.UserAction"
    method="modify">
    <result>
        /User_modify_success.jsp
    </result>
</action>
<!--调用 ThemeAction 中的 add 方法-->
<action name="Theme_add" class="com.BBS.struts2.action.ThemeAction" method="add">
    <result>
          /Theme_add_success.jsp
    </result>
</action>
<!--调用 ThemeAction 中的 list 方法-->
<action name="Theme_list" class="com.BBS.struts2.action.ThemeAction" method="list">
    <result>
        /Theme_list_success.jsp
    </result>
</action>
<!--调用 ThemeAction 中的 delete 方法-->
<action name="Theme_delete" class="com.BBS.struts2.action.ThemeAction"
    method="delete">
    <result>
        /Theme_delete_success.jsp
    </result>
</action>
<!--调用 ThemeAction 中的 modify 方法-->
<action name="Theme_modify" class="com.BBS.struts2.action.ThemeAction"
    method="modify">
    <result>
        /Theme_modify_success.jsp
```



```
    </result>
</action>
```

通过分析，发现<action>元素的 name 属性下划线前面的单词 User 与 UserAction、Theme 与 ThemeAction 的名字相同，且与物理视图文件名的第一个下划线前的单词相同；第二个单词 add、delete、modify、list 与 method 的属性值相同，且与物理视图文件名的第二个下划线前的单词相同，可以将以上 8 个逻辑 Action 配置代码改写为

```
<action name="*_*" class="com.BBS.struts2.action.{1}Action" method="{2}">
    <result>
        /{1}_{2}_success.jsp
    </result>
</action>
```

其中，表达式{1}是 name 属性值中的第 1 个 * 的值，表达式{2}是 name 属性值中的第 2 个 * 的值，以此类推。name 属性值的通配符可以使用若干个，以表达式{n}对应第 n 个 * 的值。但一般使用一个或两个 * 即可，过多的星号会使配置过于复杂。

例如，当请求为 User_list，在配置文件中先寻找是否有 name 值为 User_list 的 Action。如果没有，则查找是否有符合的通配符配置，<action name="*_*"../>元素与 User_list 符合，则表达式{1}的值为 User，表达式{2}的值为 list，则上面代码等价于以下代码。

```
<action name="User_list" class="com.BBS.struts2.action.UserAction" method="list">
    <result>
        /User_list_success.jsp
    </result>
</action>
```

要使用通配符，必须使 JSP 文件、Action 类名、方法名保持一致，包括大小写。

因此要使用通配符，首先要规定文件命名规则，如统一类名命名规则、JSP 文件命名规则、方法名命名规则等。

例如：

(1) 类名命名规则为“模块名 Action”。

(2) 方法名命名规则为：无论哪个 Action 类中，其添加方法名均为 add，删除方法均名为 delete，修改方法名均为 modify，显示列表方法名均为 list。

(3) JSP 文件命令规则为“模块名_方法名_success.jsp”。

11.5.2 实践任务 7：使用通配符调用 Action 中的指定方法

1. 任务说明

本任务以论坛管理系统的后台用户管理模块和主题管理模块为例，实践使用通配符调用 Action 中的指定方法。

当用户在后台主页(见图 11.15)单击“显示用户列表”的超链接时，打开 User_list_

success.jsp 页面，显示用户列表，如图 11.16 所示，单击“显示主题列表”的超链接时，打开 Theme_list_success.jsp 页面，显示主题列表，如图 11.17 所示。



图 11.15　后台主页



图 11.16　用户列表管理页



图 11.17　用户添加成功页

2. 任务实施

(1) 创建工程。打开工程 struts2_0200_action，复制并粘贴，命名为 BBS_Struts2_0400_ActionWildcard。

(2) 在配置文件 struts.xml 中配置 Action。打开 struts.xml，删除以下代码。

```
<!--调用 UserAction 中的 add 方法-->
<action name="User_add" class="com.BBS.struts2.action.UserAction" method="add">
    <result>
        /User_add_success.jsp
    </result>
</action>
<!--调用 UserAction 中的 list 方法-->
<action name="User_list" class="com.BBS.struts2.action.UserAction" method="list">
```

```
        <result>
            /User_list_success.jsp
        </result>
    </action>
    <!--调用 UserAction 中的 delete 方法-->
    <action name="User_delete" class="com.BBS.struts2.action.UserAction" method="delete">
        <result>
            /User_delete_success.jsp
        </result>
    </action>
    <!--调用 UserAction 中的 modify 方法-->
    <action name="User_modify" class="com.BBS.struts2.action.UserAction"
        method="modify">
        <result>
            /User_modify_success.jsp
        </result>
    </action>
    <!--调用 ThemeAction 中的 add 方法-->
    <action name="Theme_add" class="com.BBS.struts2.action.ThemeAction" method="add">
        <result>
              /Theme_add_success.jsp
        </result>
    </action>
    <!--调用 ThemeAction 中的 list 方法-->
    <action name="Theme_list" class="com.BBS.struts2.action.ThemeAction" method="list">
        <result>
            /Theme_list_success.jsp
        </result>
    </action>
    <!--调用 ThemeAction 中的 delete 方法-->
    <action name="Theme_delete" class="com.BBS.struts2.action.ThemeAction"
        method="delete">
        <result>
            /Theme_delete_success.jsp
        </result>
    </action>
    <!--调用 ThemeAction 中的 modify 方法-->
    <action name="Theme_modify" class="com.BBS.struts2.action.ThemeAction"
        method="modify">
        <result>
            /Theme_modify_success.jsp
        </result>
    </action>
```

修改 Action 的配置,代码如下:

```
<struts>
<package name="default" namespace="/main" extends="struts-default">
...
<action name="*_*" class="com.BBS.struts2.action.{1}Action" method="{2}">
```

```
    <result>
        /{1}_{2}_success.jsp
    </result>
</action>
</struts>
```

(3) 测试。部署工程,在浏览器中输入请求代码 http://localhost:8080/struts2_0200_action/main/login1,浏览器显示 index.jsp 页面,如图 11.15 所示,单击"显示用户列表"超链接,浏览器显示 User_list_success.jsp 页面,显示用户列表,如图 11.16 所示。控制台输出字符串"------调用 UserAction 类的 list 方法------",表示调用了 UserAction 类的 list 方法。

单击"显示主题列表"超链接,浏览器显示 Theme_list_success.jsp 页面,显示主题列表,如图 11.17 所示。

3. 任务总结

通过本任务,实践了如何使用通配符减少配置代码。另外,需要注意的是,使用通配符时,要首先规定文件的命名规范。

11.6 配置默认 Action

11.6.1 配置默认 Action 方法

在某些情况下,需要配置默认的 Action。

例如,当用户输入错误的请求时,在配置文件中找不到对应的 Action 处理,会显示出错信息,如图 11.18 所示。

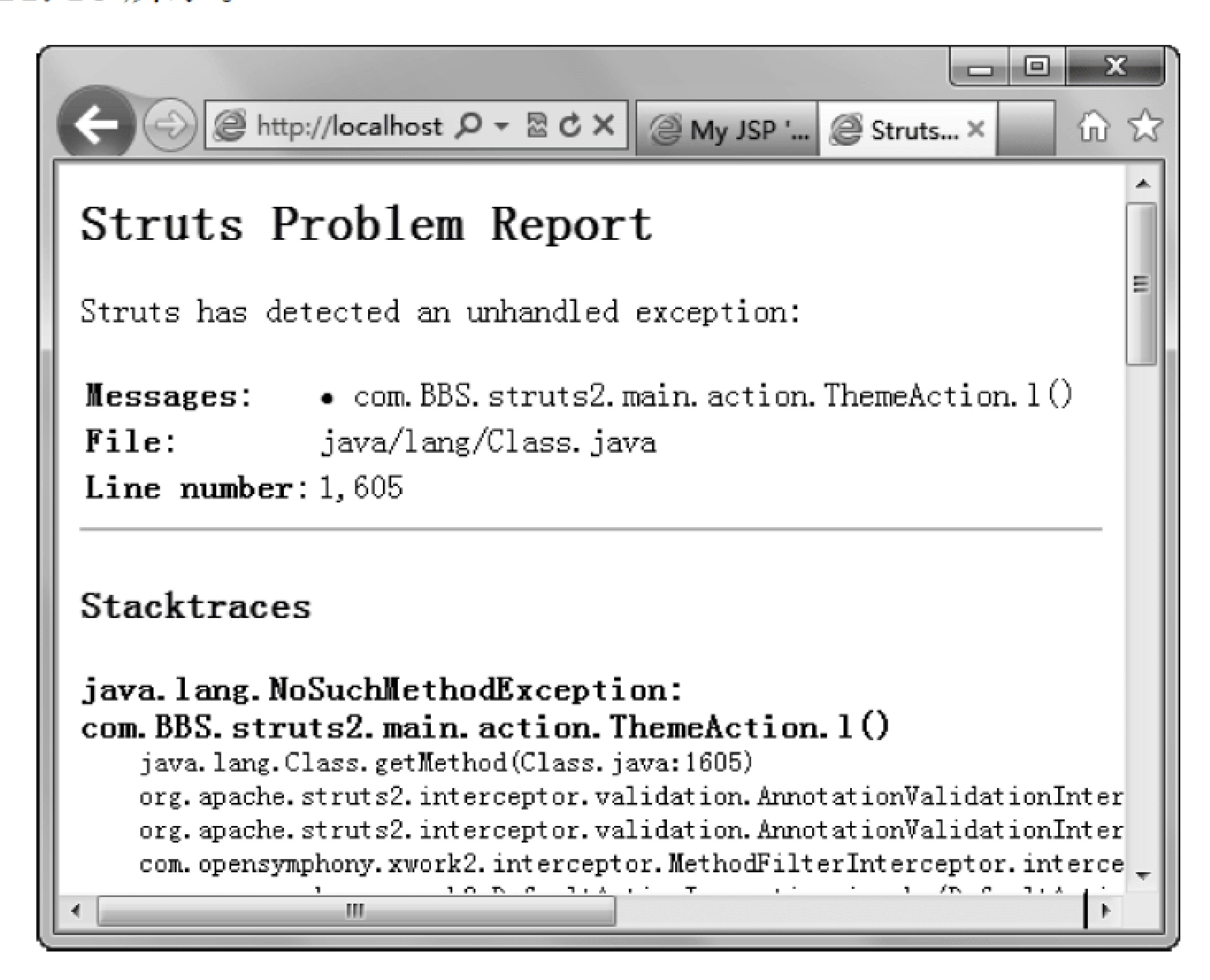


图 11.18 出错信息

出错信息用户无法理解，可以设置默认 Action，显示友好的提示信息，提示用户输入的请求有误。

使用<default-action-ref>元素配置默认 Actoin，每个<default-action-ref>元素配置一个默认 Action。下面的 struts.xml 配置片段配置了一个默认 Action。

```
<package name="default" namespace="" extends="struts-default">
    <default-action-ref name="default"></default-action-ref>
    <action name="default" >
        <result>/default.jsp</result>
    </action>
</package>
```

首先配置一个 Action，例如上面代码中 name 值为 default 的 Action，它对应的物理视图为 default.jsp。然后设置<default-action-ref>标签的 name 属性值，指定默认 Action。例如，上面代码中，设置<default-action-ref>标签的 name 属性值为 default，就设置了名为 default 的 Action 是默认 Action。另外，设置<package>标签的 namespace 属性为空，或不设置其 namespace 属性，则任何找不到匹配 namespace 的请求，都交给此<package>元素中的 Action 处理。

通过此配置，当用户输入在配置文件 struts.xml 中找不到对应的 namespace、Action 的请求时，调用默认 Action 处理请求，在浏览器显示 default.jsp 页面。

11.6.2 实践任务 8：配置默认 Action

1. 任务说明

本任务以论坛管理系统为例，实践默认 Action 设置。

当用户输入 struts.xml 中无匹配的请求时，调用默认 Action 处理请求，显示 default.jsp 页面，如图 11.19 所示。



图 11.19 default.jsp 页面

2. 任务实施

(1) 创建工程。打开工程 struts2_0200_action，复制并粘贴，命名为 BBS_Struts2_0450_ActionDefult。

(2) 配置 Action。打开配置文件 struts.xml，输入以下代码。

```
<package name="default" namespace="" extends="struts-default">
    <default-action-ref name="default"></default-action-ref>
    <action name="default" >
        <result>/default.jsp</result>
```

```
    </action>
</package>
```

(3) 测试。部署工程,在浏览器中随意输入配置文件中无匹配的请求代码,例如:

http://localhost:8080/struts2_0200_action/aaa

浏览器显示 default.jsp 页面,如图 11.19 所示。

3. 任务总结

通过本任务,实践了如何配置默认 Action。

11.6.3　实训 2:搭建论坛管理系统后台用户与版块管理模块框架

1. 实训要求

(1) 完成后台用户管理模块和版块管理模块功能。

(2) 设置默认 Action 处理配置文件中无匹配的请求,显示用户输入地址有误的提示信息。

(3) 根据书中文件命名规范命名 Action 类、jsp 文件、方法名等,使用通配符、动态方法调用等方式配置 Action。

2. 建议步骤

(1) 搭建 Struts 框架应用。

(2) 创建用户管理模块视图层文件 Login.jsp、index.jsp、User_list_success.jsp、User_delete_success.jsp、User_add.jsp、User_add_success.jsp、User_modify.jsp、User_modify_success.jsp,并编写页面提示信息等代码。

(3) 创建版块管理模块视图层文件:Theme_list_success.jsp、Theme_delete_success.jsp、Theme_add.jsp、Theme_add_success.jsp、Theme_modify.jsp、Theme_modify_success.jsp,并编写提示信息代码。

(4) 创建默认 Action 处理结果页面 default.jsp,并编写页面提示信息等代码。

(5) 实现 Action 类——UserAction,方法包括 login、list、delete、add、modify、modifyInput 等。

(6) 实现 Action 类——ThemeAction,方法包括 list、delete、add、modify、modifyInput 等。

(7) 编写配置文件 struts.xml,设置开发模式,使用通符、动态方法调用等方式配置 Action。

(8) 编写视图层文件代码,编写提交请求和显示数据的代码。

(9) 调试系统。部署工程,测试程序,如有错误,进行修改。

11.7　Action 传值方式

Action 除了处理请求,返回逻辑视图,还封装请求参数,传递来回请求的参数值。

Action 不仅可以封装用户的请求参数,还可以封装 Action 的处理结果。也就是说,

这种数据的传递不仅包括接收 JSP 页面的请求传递来的参数，也包括传递给 JSP 页面的数据。

例如用户登录系统时，Action 除了处理用户的登录请求，还接收请求的参数——用户名和密码，然后调用业务逻辑组件判断用户名和密码是否正确。

Action 传值方式常用的有属性驱动和 DomainModle 驱动。

DomainModle 的中文意思是域模型，因此 DomainModle 驱动又称为域模型驱动。

11.7.1 属性驱动

属性驱动是指通过 Action 的属性进行数据传递。

在 Struts 2 中，可以在 Action 中定义各种 Java 类型的属性，这些属性与请求的参数对应。如果请求是通过一个表单提交的，则 Action 的属性与表单中的数据相对应。Action 类通过 get * ()方法来获取值，通过 set * ()方法设置值，其中"*"代表属性名。

如请求删除某个用户，请求的参数是用户的 ID。如果用户 ID 为 int 类型，那么在 Action 中定义一个 int 类型的属性，用于接收参数，并设置属性的 get、set 方法。

代码如下：

```
import com.opensymphony.xwork2.ActionSupport;
public class UserAction extends ActionSupport{
    private int id;
    public String delete() {
        //调用业务逻辑处理删除请求的代码
        return SUCCESS;
    }
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
}
```

用户提交请求时，请求的格式如下：

```
namespace 名/Action 名?参数名=参数值
```

参数名与 Action 中的属性名一致，Action 用于接收参数的属性名为 id，在请求时，参数名也为 id。

如在 User_list_success.jsp 中，删除用户的请求代码如下：

```
<a href="main/User_delete?id=1">删除</a>
```

当用户登录系统时，请求参数是用户名和密码。用户名和密码是 String 类型，Action 定义两个 String 类型的属性，并定义其 get、set 方法，接收请求参数。

代码如下：

```
import com.opensymphony.xwork2.ActionSupport;
public class UserAction extends ActionSupport{
    private String name;
    private String password;

    public String login() {
        //调用处理登录请求的业务逻辑的代码
        return SUCCESS;
    }

    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }
}
```

当用户通过登录表单提交请求时，表单传值的格式为"属性名"，如登录页面 login.jsp 登录表单代码如下：

```
<form action="main/User_login" method="get">
    用户名：<input type="text" name="name"/><br>
    密　码：<input type="password" name="password"/><br>
    <input type="submit" value="登录">
</form>
```

用户名和密码的文本框的 name 属性值分别为"name"和"password"，与 Action 中的属性名一致。

JSP 页面还可以使用<s:property value="属性名"/>的方式，接收 Action 传递给 JSP 页面的值。

例如，登录成功页面可以显示登录的用户名，主要代码如下：

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
...
<%@taglib prefix="s" uri="/struts-tags" %>
<html>
  <head>...
  </head>
<body>
  后台主页页面 <br>
```

```
  登录的用户名为：<s:property value="name"/>
</body>
</html>
```

注意：<s:property>是 Struts 2 标签，因此要使用<%@ taglib uri="/struts-tags" prefix="s" %>进行标签声明。

关于 Struts 2 标签的内容，将在第 17 章中讲解。

另外，所有的 Struts 2 标签都以 s:开头，本书后面的示例中所有以 s:开头的标签，都需进行此标签声明，请读者留意。在测试程序时，注意添加声明，本书不再重复。

11.7.2 实践任务 9：Action 接收请求参数

1. 任务说明

本任务以论坛管理系统删除用户请求为例，实践 Action 接收请求参数。

打开 User_list_success.jsp 页面(见图 11.20)，提交删除用户的请求，Action 接收参数(用户的 id)，显示删除用户成功 User_delete_success.jsp 页面，如图 11.21 所示。



图 11.20　User_list_success.jsp



图 11.21　User_delete_success.jsp

2. 任务实施

(1) 创建工程。创建 Struts 应用，工程名为 BBS_Struts2_0500_ActionPropertyInput。

(2) 创建视图层文件。

① 创建 User_list_success.jsp。

在 WebRoot 路径下，创建 main 文件夹，在 main 文件夹下创建 User_list_success.jsp，用于显示用户信息列表，主要代码如下：

```
<body>
    用户列表管理页面 <br>
    用户 1 信息<br>
    用户 2 信息<br>
    用户 3 信息<br>
    用户 4 信息 <br>
    用户 5 信息 <br>
</body>
```

② 创建 User_delete_success.jsp。

在 main 文件夹下创建 User_delete_success.jsp，用于显示用户删除成功信息，主要代码如下：

```
<body>
    显示用户删除成功信息页面<br>
</body>
```

(3) 创建 Action。在 com.BBS.struts2.main.action 包下创建 UserAction 类，代码如下：

```
package com.BBS.struts2.main.action;
import com.opensymphony.xwork2.ActionSupport;
public class UserAction extends ActionSupport{
    private int id;
    public String delete() {
        System.out.println("id="+id);
        return SUCCESS;
    }
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
}
```

在实际开发中，“System.out.println("id="+id);”语句处应该是调用业务逻辑组件的代码，本任务主要实践 Action 如何传值，因此此处省略调用业务逻辑组件的代码，用输出语句代替，输出从页面传递给 Action 的参数的值，以验证传值是否成功。

(4) 配置 Action。打开配置文件 struts.xml，输入以下代码。

```
<package name="main" namespace="/main" extends="struts-default">
    <action name="User_delete" class="com.BBS.struts2.main.action.UserAction"
            method="delete">
        <result>
            /main/User_delete_success.jsp
        </result>
    </action>
</package>
```

(5) 编写视图层，提交请求。打开 User_list_success.jsp，在文字“用户 1 信息”后面输入删除用户的请求，代码如下：

```
<body>
    用户列表管理页面 <br>
    用户 1 信息<a href="main/User_delete?id=1">删除</a><br>
    用户 2 信息<br>
    用户 3 信息<br>
    用户 4 信息 <br>
```

```
    用户 5 信息 <br>
</body>
```

(6) 测试。部署工程，在浏览器的地址栏中输入 http://localhost:8080/BBS_Struts2_0500_ActionPropertyInput/main/User_list_success.jsp，打开 User_list_success.jsp 页面，单击“删除”超链接，显示 User_delete_success.jsp。控制台输出“id=1”。

3. 任务总结

本任务使用超链接提交带参数的请求，使用 Action 属性接收请求的参数。

11.7.3 实践任务 10：使用属性驱动方式请求参数和向 JSP 页面传值

1. 任务说明

本任务以论坛管理系统登录请求为例，实践 Action 使用属性驱动方式接收请求参数和向 jsp 页面传值。

打开 login.jsp 页面(见图 11.22)，输入用户名和密码，单击“登录”按钮，提交登录系统的请求；Action 接收参数(用户名 name 和密码 password)，判断用户名和密码是否正确，如果用户名和密码正确(用户名为 admin，密码为 123)，则跳转到 index.jsp 页面，显示登录系统成功信息(见图 11.23)，如果用户名和密码不正确，返回 login.jsp，显示登录不成功提示信息，如图 11.24 所示。



图 11.22 登录页面 login.jsp

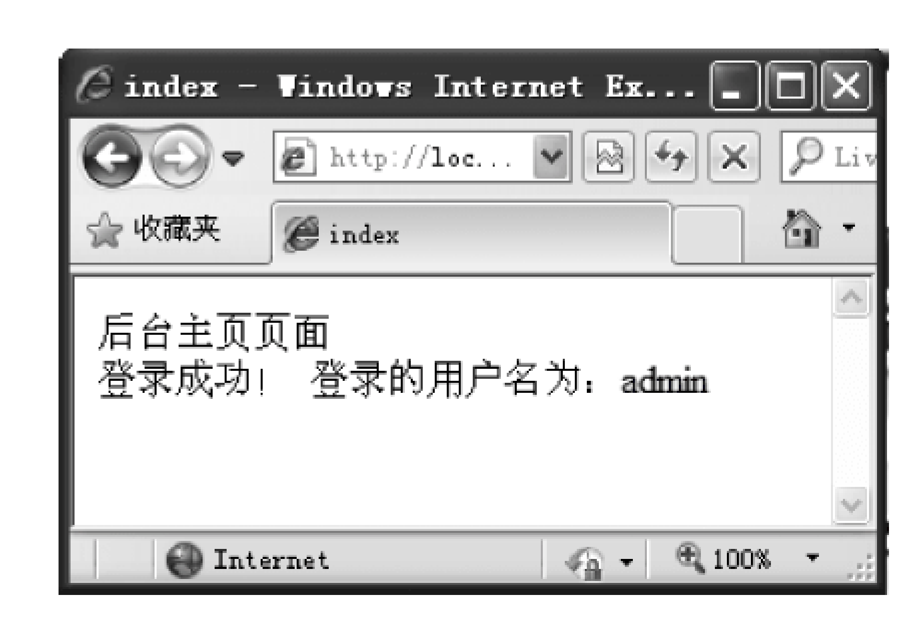


图 11.23 登录成功 index.jsp



图 11.24 登录失败 login.jsp

2. 任务实施

(1) 打开工程。打开工程 BBS_Struts2_0500_ActionPropertyInput。

(2) 创建视图层文件。

① 创建 login.jsp。在 main 文件夹下创建登录页面 login.jsp,主要代码如下:

```
<body>
   <form action="" method="get">
       用户名: <input type="text" /><br>
       密　码: <input type="password" /><br>
       <input type="submit" value="登录">
   </form>
</body>
```

② 创建 index.jsp。在 main 文件夹下创建主页 index.jsp,主要代码如下:

```
<body>
    后台主页页面<br>
</body>
```

(3) 修改 Action。修改 UserAction 类。

① 添加 login 方法,用于处理登录请求。

② 添加属性及其 get、set 方法,用于传值。定义 String 类型的属性 name 和 password,用于接收用户名和密码;定义 tip,用于向 jsp 页传递处理结果信息。

代码如下:

```
package com.BBS.struts2.main.action;
import com.opensymphony.xwork2.ActionSupport;
public class UserAction extends ActionSupport{
    ...
    private String name;
    private String password;
    private String tip;

    public String login() {
        System.out.println("name="+name);
        System.out.println("password="+password);
        if(name.equals("admin")&&password.equals("123")){
            tip="登录成功!";
            return SUCCESS;
        }else{
            tip="您输入的用户名或密码不正确,请重新输入";
            return INPUT;
        }

    }

    public String getName() {
```

```
            return name;
        }
        public void setName(String name) {
            this.name = name;
        }
        public String getPassword() {
            return password;
        }
        public void setPassword(String password) {
            this.password = password;
        }
        public String getTip() {
            return tip;
        }
        public void setTip(String tip) {
            this.tip = tip;
        }
    }
```

Action 的 login 方法应该写调用业务逻辑组件的代码，但本任务比较简单，主要练习 Action 传值，因此 login 方法直接处理了业务逻辑。login 方法根据不同的处理情况，返回不同的逻辑视图：当用户名和密码正确时，返回 SUCCESS；用户名和密码不正确时，返回 INPUT。这样做的目的是根据不同的处理结果，返回不同的逻辑视图，以对应不同的物理视图。

(4) 配置 Action。打开配置文件 struts.xml，输入以下代码。

```
<package name="main" namespace="/main" extends="struts-default">
   <action name="User_login" class="com.BBS.struts2.main.action.UserAction"
           method="login">
         <result>
             /main/index.jsp
         </result>
         <result name="input">
             /main/login.jsp
         </result>
   </action>
</package>
```

(5) 编写视图层，提交请求，设置标签的 name 属性值。打开 login.jsp，代码如下：

```
<body>
  <form action="main/User_login" method="get">
        用户名：<input type="text" name="name"/><br>
        密　码：<input type="password" name="password"/><br>
        <input type="submit" value="登录">
  </form>
</body>
```

(6) 编写视图层，设置显示数据。

① index.jsp。在 index.jsp 页面中，添加显示登录用户名和登录成功信息，代码如下：

```
<?xml version="1.0" encoding="GB18030" ?>
<%@ page language="java" contentType="text/html; charset=GB18030"
    pageEncoding="GB18030"%>
<%@ taglib uri="/struts-tags" prefix="s" %>
...
<html xmlns="http://www.w3.org/1999/xhtml">
...
<body>
  后台主页页面 <br>
  <s:property value="tip"/> 登录的用户名为: <s:property value="name"/>
</body>
```

② login.jsp。在 login.jsp 页面中，添加显示登录不成功的信息，代码如下：

```
<?xml version="1.0" encoding="GB18030" ?>
<%@ page language="java" contentType="text/html; charset=GB18030"
    pageEncoding="GB18030"%>
<%@ taglib uri="/struts-tags" prefix="s" %>
...
<html xmlns="http://www.w3.org/1999/xhtml">
...
<body>
  <s:property value="tip"/>
   <form action="main/User_login" method="get">
...
</body>
```

(7) 测试。部署工程，在浏览器的地址栏中输入 http://localhost:8080/BBS_Struts2_0500_ActionPropertyInput/main/login.jsp，打开 login.jsp 页面，输入用户名为 aaa、密码为 111，单击“登录”按钮，返回 login.jsp 页面，显示提示信息“您输入的用户名或密码不正确，请重新输入”，如图 11.23 所示。

控制台输出如下：

```
name=aaa
password=111
```

重新输入用户名为 admin，密码为 123，单击“登录”按钮，打开 index.jsp 页面，显示登录用户名和“登录成功！”提示信息，如图 11.24 所示。

控制台输出如下：

```
name=admin
password=123
```

3. 任务总结

本任务使用表单提交请求并传递参数，使用 Action 属性接收请求和参数，并根据处理结果的不同，返回不同的逻辑视图，并封装处理结果的数据。

11.7.4 域模型驱动

所谓域模型驱动是指 Action 通过 JavaBean 模型进行数据传递。采用这种方式，在 Action 类中定义 JavaBean 的对象，JavaBean 所封装的属性与请求中的参数对应，JavaBean 作为数据传递的载体。

采用域模型方式，一般通过表单提交请求，则 JavaBean 所封装的属性与表单的数据相对应。使用域模型驱动方式，Action 类通过 get * ()方法来获得模型、set * ()方法来设置模型中属性的值，其中"*"代表具体的模型对象。

例如，实践任务 10 中的系统登录就可以采用模型驱动的方式。定义一个域模型类 User 用来封装 name 和 password，该模型类代码如下：

```
package com.BBS.struts2.model;
public class User {
    private String name;
    private String password;
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }
}
```

Action 类中不再定义 name 和 password 属性，而定义一个 User 类型的域模型，并定义其 get * ()和 set * ()方法，代码如下：

```
package com.BBS.struts2.main.action;

import com.BBS.struts2.model.User;
import com.opensymphony.xwork2.ActionSupport;

public class User_Action extends ActionSupport{
    private User user;

    public String login() {
```

```
            //调用业务逻辑处理删除请求的代码
            return "success";
        }

        public User getUser() {
            return user;
        }
        public void setUser(User user) {
            this.user = user;
        }
    }
```

当用户通过登录表单提交请求时，表单传值的格式为“对象名.属性名”，如登录页面 login.jsp 中登录表单的代码如下：

```
<form action="main/User_login" method="get">
    用户名：<input type="text" name="user.name"/><br>
    密　码：<input type="password" name="user.password"/><br>
    <input type="submit" value="登录">
</form>
```

用户名和密码的文本框的 name 属性值分别为"user.name"和"user.password"，与 Action 中的域模型一致。

当 JSP 页面从 Action 中取值时，取值格式也必须为“对象名.属性名”，例如在主页显示登录系统的用户名，使用<s:property value="user.name"/>取得用户名。

11.7.5　实践任务 11：使用域模型驱动方式接收请求参数

1. 任务说明

本任务以论坛管理系统登录请求为例，实践 Action 使用域模型驱动方式接收请求参数。

打开 login.jsp 页面(见图 11.25)，输入用户名和密码，单击“登录”按钮，提交登录系统的请求；Action 接收参数(用户名 name 和密码 password)，跳转到 index.jsp 页面，显示登录用户名，如图 11.26 所示。



图 11.25　登录页面 login.jsp



图 11.26　登录成功 index.jsp

2. 任务实施

(1) 创建工程。创建 Struts 应用 BBS_Struts2_0600_ActionDomainModeInput。

(2) 创建视图层文件。

① 创建 login.jsp。在 WebRoot 路径下，创建 main 文件夹，在 main 文件夹下创建登录页面 login.jsp，主要代码如下：

```
<body>
    <form action="" method="get">
        用户名: <input type="text" /><br>
        密  码: <input type="password" /><br>
        <input type="submit" value="登录">
    </form>
</body>
```

② 创建 index.jsp。在 main 文件夹下创建主页 index.jsp，主要代码如下：

```
<body>
    后台主页页面<br>
</body>
```

(3) 创建 JavaBean。在 com.BBS.struts2.model 包下创建 User 类，包括 name 和 password 属性。代码如下：

```
package com.BBS.struts2.model;

public class User {
    private String name;
    private String password;
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }
}
```

(4) 创建 Action。在 com.BBS.struts2.main.action 包下创建 UserAction 类。

① 添加 login 方法，用于处理登录请求。

② 定义 User 类型的域模型及其 get、set 方法。

代码如下：

```
package com.BBS.struts2.main.action;

import com.BBS.struts2.model.User;
import com.opensymphony.xwork2.ActionSupport;

public class UserAction extends ActionSupport{

    private User user;

    public String login() {
        System.out.println("name="+user.getName());
        System.out.println("password="+user.getPassword());
        return "success";
    }

    public User getUser() {
        return user;
    }

    public void setUser(User user) {
        this.user = user;
    }
}
```

Action 的 login 方法应该写调用业务逻辑组件的代码，但本任务比较简单，主要练习 Action 传值，因此 login 方法处使用输出语句，输出 user 对象的属性值，检验参数传递是否成功。

(5) 配置 Action。打开配置文件 struts.xml，输入以下代码。

```
<package name="main" namespace="/main" extends="struts-default">
   <action name="User_login" class="com.BBS.struts2.main.action.UserAction"
           method="login">
        <result>
            /main/index.jsp
        </result>
   </action>
</package>
```

(6) 编写视图层，提交请求，设置标签的 name 属性值。打开 login.jsp，代码如下：

```
<body>
  <form action="main/User_login" method="get">
       用户名：<input type="text" name="user.name"/><br>
       密　码：<input type="password" name="user.password"/><br>
       <input type="submit" value="登录">
```

```
  </form>
</body>
```

注意：表单元素的 name 属性值是“对象名.属性名”。

（7）编写视图层，设置显示数据。在 index.jsp 页面中，显示登录用户名的代码如下：

```
<body>
  后台主页页面 <br>
  欢迎<s:property value="user.name"/>登录系统!
</body>
```

（8）测试。部署工程，在浏览器的地址栏中输入 http://localhost:8080/BBS_Struts2_0600_ActionDomainModeInput/main/login.jsp，打开 login.jsp 页面，输入用户名为 aaa、密码为 111，单击“登录”按钮，打开 index.jsp 页面，显示登录用户名和“登录成功!”提示信息和登录用户名。

控制台输出如下：

```
name=aaa
password=111
```

3. 任务总结

本任务使用表单提交请求并传递参数，使用域模型驱动方式接收请求和参数 user.name、user.password，并使用域模型向 jsp 页面传递数据 user.name。

11.7.6 拓展任务

实现后台用户管理模块中创建用户的 Action 传值。

建议步骤：

（1）打开工程——BBS_Struts2_0600_ActionDomainModeInput。

（2）创建视图层文件 User_add_input.jsp 和 User_add_success.jsp，并编写页面提示信息等代码。

（3）创建域模型 UserDTO 类，包括用户名、密码、密码确认等属性。

（4）编写 UserAction 类，实现参数传递。

（5）编写 struts.xml 文件，配置 Action。

（6）打开 User_add_input.jsp 文件编写添加用户的请求，表单元素的 name 属性值。

（7）部署工程。

（8）调试系统。打开浏览器，输入客户端请求，测试程序，如有错误，进行修改。

11.7.7 实训 3：实现论坛管理系统后台用户与版块管理模块 Action 传值

完成后台用户管理模块和版块管理模块 Action 传值。

11.8　本章小结

本章详细讲解了 Struts 2 应用核心 Action，包括：Action 类的作用及其实现方法；在 Action 中定义多个方法时，如何调用指定方法；通配符的使用方法，使用通配符的好处和注意事项；默认 Action 的配置，Action 采用属性驱动和域模型驱动实现数据的传递。

另外，为方便练习 Action，在讲解 Action 之前，介绍了开发模式的设置及其作用。

第12章

Struts 2 的类型转换器

第 11 章介绍了 Struts 2 框架如何从表现层接收参数并传递给 Action。在此过程中，Web 应用的客户端输入的数据均为字符串类型，而传递给服务器端的 Action 的数据可能是各种各样的数据类型，这就造成 B/S 两端的类型不兼容问题。本章重点讲解 Struts 2 框架如何将客户端的字符串类型转换为服务器端所需要的类型。

本章要点：

- Struts 2 类型转换的原理
- Struts 2 内置类型转换器
- 简单数据类型的转换
- 引用类型的转换
- 集合的类型转换
- 类型转换错误的处理

12.1 类型转换概述

在 Web 应用中，客户端所输入的数据都是 String 类型，在传递给服务器端后，需要经过类型转换，转换成服务器端处理中需要的各种数据类型。

Struts 2 是一个成熟的 MVC 框架，它提供了类型转换的功能，可以自动处理 String 类型与常用类型之间的转换。

12.2 Struts 2 内置类型转换器

12.2.1 内置类型转换器简介

Struts 2 提供了强有力的表现层类型转换机制，不需要开发人员过多干预即可自动完成转换，这些类型转换工作都是由 Struts 2 框架的内置类型转换器完成的。

Struts 2 内置了字符串类型和以下类型之间的类型转换器。

(1) boolean 和 Boolean：字符串和布尔值之间的转换。

(2) char 和 Character：字符串和字符之间的转换。

(3) int 和 Interger：字符串和整型值之间的转换。

(4) long 和 Long：字符串和长整型值之间的转换。

(5) float 和 Float：字符串和单精度浮点值之间的转换。

(6) double 和 Double：字符串和双精度浮点值之间的转换。

(7) Date：字符串和日期类型之间的转换，其中日期格式使用用户请求所在 Locale 的 SHORT 格式。

(8) 数组：在默认情况下，数据元素是字符串。

(9) 集合：在默认情况下，假定集合元素类型为 String，并创建一个新 ArrayList 封装所有的字符串。

一般来说，Struts 为我们提供的内置转换器能实现大部分需求，不需要定义新的转换器。

例如，注册用户时，注册客户端收集用户输入的注册信息，包括姓名（字符串类型）、年龄（整型）和生日信息（日期类型）。如图 12.1 所示，其步骤如下。



图 12.1　注册信息的收集页面

(1) 其注册信息收集页面代码如下：

```
<body>
<h1>请输入您的注册信息</h1>
    <form action="conversion/con" method="post">
       姓名：<input type="text" name="name"/><br>
       年 龄：<input type="text" name="age"/><br>
       生日：<input type="text" name="birthday"/><br>
       <input type="submit" value="添加">
     </form>
</body>
```

(2) 创建 Action，接收注册信息，代码如下：

```
package com.cvit.struts2.action;
import java.util.Date;
import com.opensymphony.xwork2.ActionSupport;
public class ConversionAction   extends ActionSupport{

    private String name;
    private int age;
    private Date birthday;

    public String execute(){
        System.out.println("name="+name);
        System.out.println("age="+age);
```

```
            System.out.println("birthday ="+birthday);
            return SUCCESS;
        }
        //属性的 getter 和 setter 方法略
        ...
}
```

(3) 在 struts.xml 文件中编写配置，主要代码如下：

```
<package name="conversion" extends="struts-default" namespace="/conversion">
    <action name="con" class="com.cvit.struts2.action.ConversionAction">
        <result>/index.jsp</result>
    </action>
</package>
```

(4) 编写 index.jsp，显示数据类型转换后的注册信息。

代码如下：

```
<body>
    注册成功!<br/>
    类型转换结果: <br/>
    姓名: <s:property value="name"/><br/>
    年龄: <s:property value="age"/><br/>
    生日: <s:property value="birthday"/><br/>
</body>
```

(5) 当输入如图 12.2 所示的用户信息，提交表单，控制台输出信息如下：

```
name=Tom
age=20
birthday =Mon Jan 01 00:00:00 CST 1990
```

在浏览器中显示 index.jsp 页面，如图 12.3 所示。




图 12.2 输入注册信息

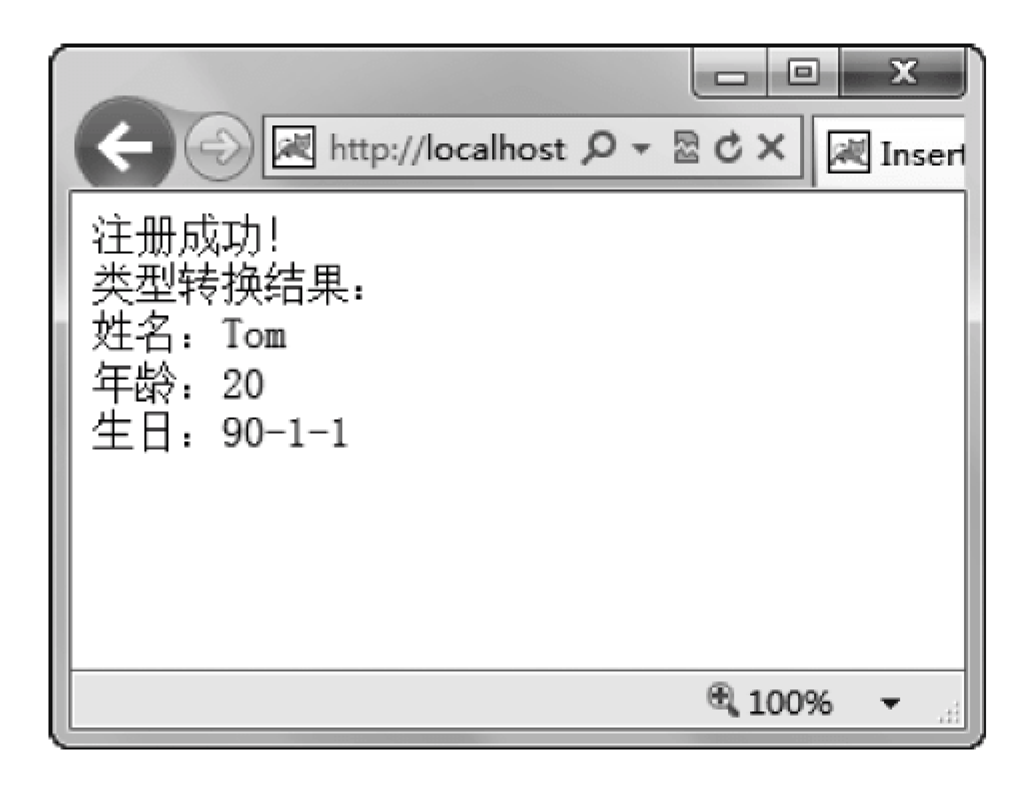


图 12.3 显示类型转换后数据的页面

数据类型自动实现了转换。

如果对生日显示的信息格式不满意，可以使用 Struts 2 标签 s:date 设置。

格式如下：

```
<s:date name="birthday" format="日期格式"/>
```

如在 JSP 页面中输入代码<s:date name="birthday" format="yyyy-MM-dd"/>，则浏览器显示 1990-01-01；如果代码为<s:date name="birthday" format="yyyy 年 MM 月 dd 日"/>，则浏览器显示 1990 年 01 月 01 日；如果代码为<s:date name="birthday" format="yyyy 年 MM 月 dd 日 HH:mm:ss"/>，则浏览器显示 1990 年 01 月 01 日 00:00:00，表示 1990 年 01 月 01 日 00 时 00 分 00 秒。

12.2.2　实践任务 1：Struts 2 内置类型转换器转换简单数据类型

1. 任务说明

本任务以论坛管理系统用户注册信息为例，实践 Struts 2 内置类型转换器转换简单数据类型。

打开 User_add_input.jsp 页面(见图 12.1)，输入姓名、年龄、出生日期(见图 12.2)，单击“添加”按钮，跳转到 index.jsp 页面，显示转换后的数据信息，如图 12.4 所示。

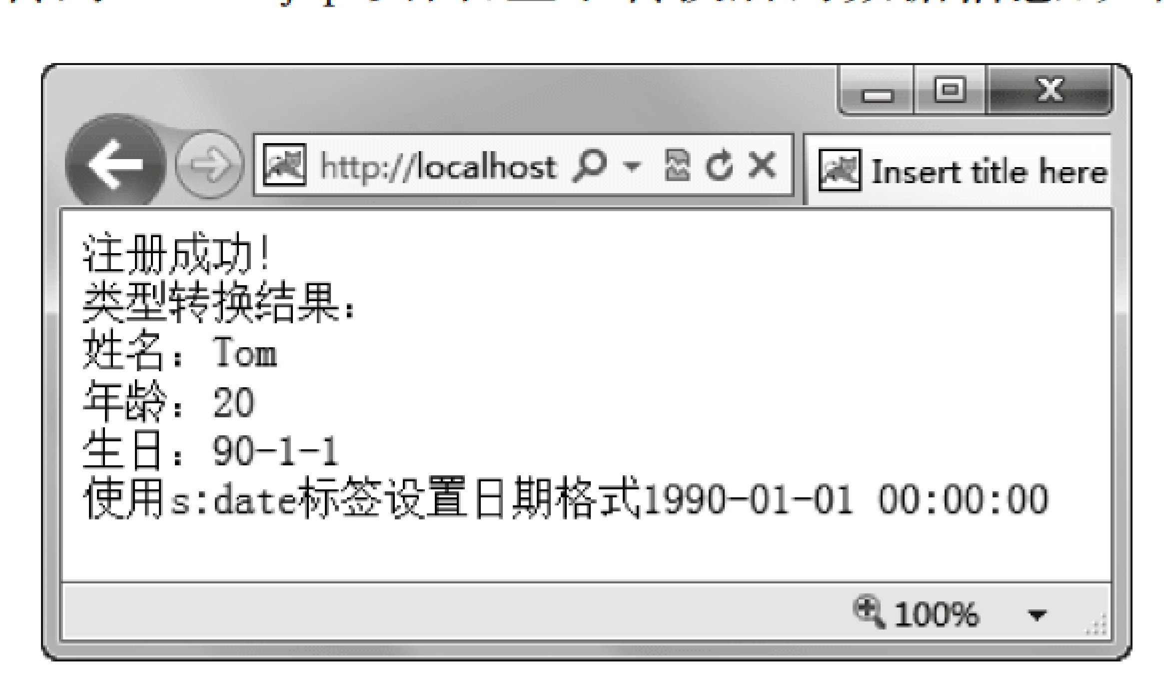


图 12.4　index.jsp 页面

2. 任务实施

(1) 创建工程。创建 Struts 应用 BBS_Struts2_0700_typeConversion。

(2) 创建视图层文件。

① 创建 User_add_input.jsp。在 WebRoot 路径下创建 User_add_input.jsp，主要代码如下：

```
<body>
<h1>请输入您的注册信息</h1>
    <form action=" " method="post">
        姓名: <input type="text"/><br>
        年龄: <input type="text"/><br>
        生日: <input type="text" /><br>
        <input type="submit" value="添加">
    </form>
</body>
```

② 创建 index.jsp。主要代码如下：

```
<body>
    注册成功!<br/>
    类型转换结果：<br/>
    姓名：<br/>
    年龄：<br/>
    生日：<br/>
</body>
```

(3) 创建 Action。在 com.BBS.struts2.action 包下创建 ConversionAction 类，代码如下：

```
package com.cvit.struts2.action;
import java.util.Date;
import com.opensymphony.xwork2.ActionSupport;
public class ConversionAction  extends ActionSupport{
    private String name;
    private int age;
    private Date birthday;

    public String execute(){
        System.out.println("name="+name);
        System.out.println("age="+age);
        System.out.println("date="+date);
        return SUCCESS;
    }
    //属性的 getter 和 setter 方法略
    ...
}
```

(4) 配置 Action。打开配置文件 struts.xml，输入以下代码。

```
<package name="conversion" extends="struts-default" namespace="/conversion">
    <action name="con" class="com.cvit.struts2.action.ConversionAction">
        <result>/index.jsp</result>
    </action>
</package>
```

(5) 编写视图层，提交请求，设置标签的 name 属性值。打开 User_add_input.jsp，代码如下：

```
<body>
<h1>请输入您的注册信息</h1>
    <form action="conversion/con" method="post">
       姓名：<input type="text" name="name"/><br>
       年龄：<input type="text" name="age"/><br>
       生日：<input type="text" name="birthday"/><br>
       <input type="submit" value="添加">
     </form>
```

```
</body>
```

(6) 编写视图层,设置显示数据。在 index.jsp 页面中显示转换到的数据,代码如下:

```
<body>
    注册成功!<br/>
    类型转换结果:<br/>
    姓名:<s:property value="name"/><br/>
    年龄:<s:property value="age"/><br/>
    生日:<s:property value="birthday"/><br/>
    使用s:date标签设置日期格式<s:date name="birthday"
        format="yyyy-MM-dd HH:mm:ss"/>
</body>
```

(7) 测试。部署工程,在浏览器的地址栏中输入 http://localhost:8080/BBS_Struts2_0700_typeConversion/User_add_input.jsp,打开/User_add_input.jsp 页面,输入姓名为 Tom、年龄为 20、出生日期为 1990-01-01(见图 12.2),单击"添加"按钮,打开 index.jsp 页面,显示转换后的数据信息(见图 12.4)。

控制台输出如下:

```
name=Tom
age=20
birthday =Mon Jan 01 00:00:00 CST 1990
```

3. 任务总结

本任务实践 Struts 2 内置类型转换器自动实现简单数据类型的转换。

12.3　引用类型的转换方式

12.3.1　简介

Struts 2 不但可以转换简单数据类型,当页面<input>标签的 name 属性值与 Action 中的引用类型"对象名.属性名"相对应时,Struts 2 还可以自动将页面中的请求参数自动转换为引用类型的对象。

例如,将实践任务 1 中的注册表单代码作如下修改。

```
<form action="conversion/con" method="post">
    姓名:<input type="text" name="user.name"/><br>
    年龄:<input type="text" name="user.age"/><br>
    生日:<input type="text" name="user.birthday"/><br>
    <input type="submit" value="添加">
</form>
```

此时,Action 的属性为 User 类型的对象,其 User 类的代码如下:

```
package com.cvit.struts2.model;
```

```
import java.util.Date;
public class User {
    private String name;
    private int age;
    private Date birthday;
    //属性的 getter 和 setter 方法略
    ...
}
```

Action 类的主要代码如下：

```
public class ConversionAction  extends ActionSupport{
    private User user;
    public String execute(){
        System.out.println("name="+user.getName());
        System.out.println("age="+user.getAge());
        System.out.println("birthday ="+user.getBirthday());
        return SUCCESS;
    }

    public User getUser() {
        return user;
    }
    public void setUser(User user) {
        this.user = user;
    }
}
```

其运行结果与实践任务 1 一致，这是因为 Struts 2 的内置类型转换器将参数 user.name、user.age 和 user.birthday 转换为 User 类型对象 user 的属性 name、age 和 birthday。

12.3.2 实践任务 2：Struts 2 内置类型转换器转换引用数据类型

1. 任务说明

本任务以论坛管理系统用户注册信息为例，实践 Struts 2 内置类型转换器转换引用数据类型。

打开 User_add_input.jsp 页面(见图 12.1)，输入姓名、年龄、出生日期(见图 12.2)，单击“添加”按钮，跳转到 index.jsp 页面，显示转换后的数据信息，如图 12.4 所示。

2. 任务实施

(1) 创建工程。将应用 BBS_Struts2_0700_typeConversion 复制并粘贴，将其改名为 BBS_Struts2_0750_typeConversionObject。

(2) 修改视图层文件。

① 修改 User_add_input.jsp。在 WebRoot 路径下创建 User_add_input.jsp，主要代码如下：

```
<body>
<h1>请输入您的注册信息</h1>
    <form action="conversion/con" method="post">
        姓名：<input type="text" name="user.name"/><br>
        年龄：<input type="text" name="user.age"/><br>
        生日：<input type="text" name="user.birthday"/><br>
        <input type="submit" value="添加">
    </form>
</body>
```

② 修改 index.jsp。主要代码如下：

```
<body>
    注册成功!<br/>
    类型转换结果：<br/>
    姓名：<s:property value="user.name"/><br/>
    年龄：<s:property value="user.age"/><br/>
    生日：<s:property value="user.birthday"/><br/>
    使用 s:date 标签设置日期格式<s:date name="user.birthday"
                                    format="yyyy-MM-dd HH:mm:ss"/>
</body>
```

(3) 创建 User 类。代码如下：

```
package com.cvit.struts2.model;

import java.util.Date;
import java.util.List;

public class User {
    private String name;
    private int age;
    private Date birthday;
    //属性的 getter 和 setter 方法略
    ...
}
```

(4) 修改 Action。代码如下：

```
package com.cvit.struts2.action;
import java.util.Date;
import com.opensymphony.xwork2.ActionSupport;
public class ConversionAction  extends ActionSupport{
    private User user;
    public String execute(){
        System.out.println("name="+user.getName());
        System.out.println("age="+user.getAge());
        System.out.println("birthday="+user.getBirthday());
        System.out.println("interest"+user.getInterest().size());
        return SUCCESS;
```

```
        }

        public User getUser() {
            return user;
        }
        public void setUser(User user) {
            this.user = user;
        }
    }
```

(5) 测试。部署工程,在浏览器的地址栏中输入 http://localhost:8080/BBS_Struts2_0750_typeConversionObject/User_add_input.jsp,打开/User_add_input.jsp 页面,输入姓名为 Tom、年龄为 20、出生日期为 1990-01-01(见图 12.2),单击"添加"按钮,打开 index.jsp 页面,显示转换后的数据信息(见图 12.4)。

控制台输出如下:

```
name=Tom
age=20
birthday =Mon Jan 01 00:00:00 CST 1990
```

3. 任务总结

本任务实践 Struts 2 内置类型转换器自动实现引用数据类型的转换。

12.4 集合类型的转换方式

Struts 2 还可以处理集合类型的数据转换。比如,在页面中的复选按钮,会提交多个同名的参数及其相应的不同参数值,Struts 2 的内置校验器会自动将其转换为集合类型的数据。

例如用户注册信息时,选择多种爱好,Struts 2 的内置校验器会自动将多个爱好转换为一个集合,每个爱好是集合中的一个元素。

其实现步骤如下。

(1) 在 User 类中创建 interest 集合属性,用于保存多个爱好,代码如下:

```
package com.cvit.struts2.model;
import java.util.Date;
import java.util.List;

public class User {
    private String name;
    private int age;
    private Date birthday;
    private List interest;
    //name、age、birthday 的 getter 和 setter 方法略
    ...
    public List getInterest() {
```

```
        return interest;
    }
    public void setInterest(List interest) {
        this.interest = interest;
    }
}
```

(2) Action 中的测试代码如下：

```
public class ConversionAction  extends ActionSupport{
    private User user;

    public String execute(){
        System.out.println("name="+user.getName());
        System.out.println("age="+user.getAge());
        System.out.println("birthday="+user.getBirthday());
        System.out.println("interest"+user.getInterest().size());
        return SUCCESS;
    }

    public User getUser() {
        return user;
    }
    public void setUser(User user) {
        this.user = user;
    }
}
```

(3) 在客户端提交的请求如下：

```
http://localhost:8080/BBS_Struts2_0750_typeConversionObject/conversion/con?user.interest=
chinese&user.interest=englist& user.interest=math
```

控制台输出信息如下：

```
name=null
age=0
birthday=null
interest 集合的元素个数为 3
user.getInterest().size()计算集合 interest 的元素个数
```

同时我们也可以注意到，当不输入姓名、年龄、生日信息时，Struts 2 框架会根据其数据类型，自动设置默认值。

12.5　类型转换的错误处理

如果用户输入的数据有误，导致类型转换失败，Struts 2 会调用处理类型转换错误的拦截器“conversionError”，并将异常信息封装在一个 fieldError 中，返回逻辑视图 input。

注意：关于拦截器的内容将在第 18 章中详细介绍。

<s:fielderror>标签可以输出 fieldError 中的异常信息。

要查看 Struts 框架在类型转换错误时保存的信息，操作步骤如下。

(1) 继承 ActionSupport 类：如果想在页面上输出类型转换时发生的错误，处理表单请求的 Action 类必须继承 ActionSupport 类。

(2) 配置物理视图：在 struts.xml 中配置类型转换错误时，对应的物理视图，代码如下：

```
<action name="con" class="com.cvit.struts2.action.ConversionAction">
  <result>/index.jsp</result>
  <result name="input">/User_add_input.jsp</result>
</action>
```

(3) 插入<s:fielderror>标签。打开 User_add_input.jsp，插入<s:fielderror>标签，代码如下：

```
<body>
    <h1>请输入您的注册信息</h1>
    <s:fielderror/>
    <form action="conversion/con" method="post">
        姓名：<input type="text" name="user.name"/><br>
        年龄：<input type="text" name="user.age"/><br>
        生日：<input type="text" name="user.birthday"/><br>
        <input type="submit" value="添加">
   </form>
</body>
```

当输入年龄不是整数时，例如输入“uuu”，如图 12.5 所示，则 conversionError 处理类型转换异常，<s:fielderror>标签显示类型转换异常信息，如图 12.6 所示。



图 12.5　输入错误的年龄

异常信息是英文，如果想显示中文或其他内容，可以使用类型校验器对类型转换进行校验。数据输入的校验在第 13 章详细讲解。



图 12.6　输入年龄有误的错误提示信息

12.6　本章小结

本章主要介绍了 Struts 2 的数据类型转换机制，从视图层所涉及的类型转换处理讲起，介绍了类型转换的作用和意义，接下来详细讲解了 Struts 2 内置类型转换器对简单数据和引用数据类型转换的实现，还介绍了集合类型的转换和转换异常处理机制。

第13章

Struts 2 的输入校验

输入校验在 Web 应用中具有重要作用，因为 Web 应用的开放性，网络上所有的浏览者都可以使用 Web 应用，因此 Web 应用通过输入页面接收的数据是非常复杂的，不仅会包含正常用户的误输入，还可能包含恶意用户的恶意输入。一个稳健的应用系统必须将这些非法的输入阻止在应用之外，使服务器端程序不会接收到非法数据，从而保证 Web 应用安全顺序的执行。

Struts 2 框架提供了非常强大的输入校验体系，本章讲解 Struts 2 框架中输入校验的实现原理和实现过程。

本章要点：

- 输入校验的作用和内容
- 客户端输入校验
- 如何使用编程方式实现 Struts 2 输入校验
- validate()方法的使用
- validateXxx()方法的使用
- 常用内建校验器的使用

13.1　输入校验概述

异常的输入，可能会导致系统非正常中断，甚至可能会导致系统崩溃。因此应用程序必须能将异常数据拦截在服务器之外，通常的做法是当遇到异常输入时，应用程序直接返回到表现层，提示用户必须重新输入，从而过滤掉异常输入。这种对异常输入的过滤，就是输入校验，也称为数据校验或数据输入校验。

输入校验是指在数据提交给程序之前，对数据的合法性进行检查，只允许合法的数据进入应用程序。输入校验分为客户端校验和服务器校验，客户端校验主要用于过滤正常用户的误输入，主要通过 JavaScript 代码完成，服务器端校验是整个 Web 应用阻止非法数据进入应用的最后一道防线，主要通过在应用中编程实现。

13.1.1　客户端校验

客户端校验就是使用 JavaScript 语言在数据收集页面(通常是表单输入页)编写程序，过滤输入的数据。JavaScript 语言是一个功能非常强大的脚本语言，

代码支持正则表达式，可以在数据提交给服务器之间，对数据进行验证。

例如，用户登录系统时，需要输入用户名和密码。其输入的用户名和密码必须是非空的字符串。登录表单代码如下：

```
<form  name="loginFrm"  action=""  method="get">
    用户名：<input  type="text"  name="username"/>
    密  码：<input  type="password"  name="password"/>
    <input  type="submit"  value="登录"/>
</form>
```

JavaScript 校验代码如下：

```
<script type="text/javascript">
function validate(){
    //定义错误提示字符串
    var errStr="";
    var username=trim(document.loginFrm.username.value);
    var password=trim(document.loginFrm.password.value);
    //判断用户名不能为空
  if(username==""||username==null){
      errStr+="您没有输入用户名，请重新输入";
  }
  //判断密码不能为空
  if(password ==""||password==null){
      errStr+="\n 您没有输入密码，请重新输入";
  }
  //如果 errStr 为空，表明通过客户端校验
  if(errStr==""){
      return true;
  }
  //没有通过客户端输入校验，弹出提示对话框显示错误提示信息
  else{
      alert(errStr);
      return false;
  }
}
</script>
```

定义了上面的客户端校验函数后，还需要在页面的表单中调用该函数才能进行客户端校验。可以通过将校验函数绑定到表单中的 onsubmit 函数的方法调用。

将表单元素修改，代码如下：

```
<form  name="loginFrm"  action=""  method="get"  onsubmit="return validate();">
    用户名：<input  type="text"  name="username"/>
    密  码：<input  type="password"  name="password"/>
    <input  type="submit"  value="登录"/>
</form>
```

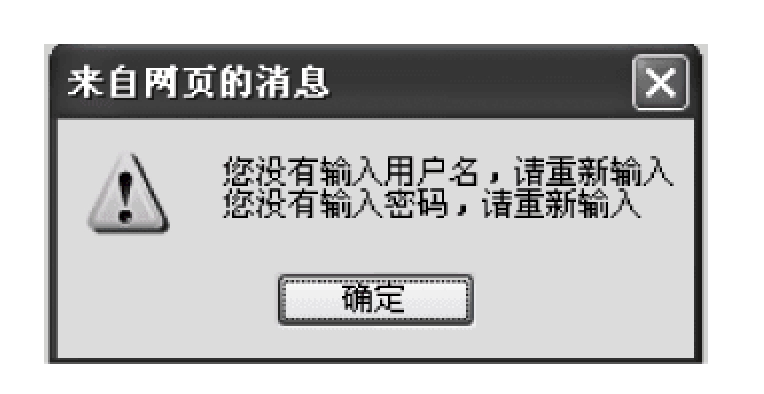


图 13.1 客户端校验的效果

增加了以上的客户端校验后，如果未输入用户名和密码，系统将弹出对话框，提示用户输入数据中有误码，如图 13.1 所示。

上面页面添加了客户端校验，当用户输入的数据不满足校验要求，系统不会将数据提交给服务器，而是返回视图层，提示用户重新输入。

13.1.2 服务器端输入校验

客户端校验主要用于验证正常用户的误操作，而如果是恶意用户，会轻易绕开客户端校验，因此必须设置服务器校验，过滤异常数据。

服务器输入校验可以通过使用编程的方式实现和使用内建校验器的方式实现。

13.2 手动完成输入校验

手动完成输入校验的方式就是通过将校验数据的代码编写在 Action 类中来实现输入校验。实现起来很简单，下面我们一起来看看在 Struts 2 框架中，如何手动完成输入校验。

13.2.1 在 Action 处理请求的方法中直接实现输入校验

例如，处理用户登录请求的方法是 login()方法，可以将实现登录输入校验的代码写在 login()方法中。当 Action 执行 login()方法时，对应的校验代码也被执行，如果输入的数据有误，可以通过 addFieldError()或 addActionError()方法添加数据输入错误的提示信息，在 JSP 页中也可以获取提示信息。

例如，登录表单代码如下：

```
<form action="user/User!login" method="get">
    用户名：<input type="text" name=" user.name"/><br>
    密  码：<input type="password" name="user. password"/><br>
    <input type="submit" value="登录">
</form>
```

则实现输入校验的步骤如下。

(1) 在 Action 的 login()方法中，加入以下校验代码。

```
public class UserAction extends ActionSupport{
  private User user;

  public String login() {
  //校验代码
      if(user.getName()==null||user.getName().equals("")){
```

```
            this.addFieldError("name", "name can not be null");
            return INPUT;
        }
        if(user.getPassword()==null||user.getPassword().equals("")){
            this.addFieldError("password", "password can not be null");
            return INPUT;
        }

    }
  //业务逻辑代码,处理请求
  ...
  return "success";
  ...
}
```

在以上的代码中，输入校验代码的作用是：当用户名或密码为空时，使用 addFieldError()方法，将错误提示信息"name can not be null"赋值给"name"、"password can not be null"赋值给"password"，并将"name"和"password"放到 fieldErrors 中，然后返回字符串常量 INPUT，login()方法中的业务逻辑处理代码不执行。

如果用户名和密码都不为空时，执行业务逻辑代码，检验用户名和密码是否正确，本例中，此段代码省略。

（2）在 struts.xml 中配置 Action，主要代码如下：

```
<package name="user" namespace="/user" extends="struts-default">
  <action name="User " class="com.BBS.struts2.main.action.UserAction" >
    <result>
          /main/index.jsp
    </result>
    <result name="input">/main/login.jsp</result>
  </action>
</package>
```

（3）在登录页面显示错误提示信息，主要代码如下：

```
<form action="user/User!login" method="get">
   用户名：<input type="text" name="user.name"/><br>
   密　码：<input type="password" name="user.password"/><br>
   <input type="submit" value="登录">
   <s:fielderror fieldName="name" theme="simple"/>
   <s:fielderror fieldName="password" theme="simple"/>
   <s:property value="errors.name[0]"/>
</form>
```

<s:fielderror>标签是 Struts 框架标签，可以通过 fieldName 属性，读取 fieldErrors 中的值。

当用户输入空的用户名并提交请求后，Action 的 login()方法将错误提示信息"name can not be null"赋值给 name，并将 name 放到 fieldErrors 中，不执行业务逻辑代码，直接返回字符串常量 INPUT。则界面显示登录页面，并显示 fieldErrors 中的错误提示信息，

如图 13.2 所示。

用户名不为空，密码为空时，Action 的 login()方法将错误提示信息"password can not be null"赋值给 password，并将 password 放到 fieldErrors 中，不执行业务逻辑代码，直接返回字符串常量 INPUT。则界面显示登录页面，并显示 fieldErrors 中的错误提示信息，如图 13.3 所示。



图 13.2　输入空的用户名并提交后的页面



图 13.3　输入空的密码并提交后的页面

13.2.2　Struts 2 中的值栈

值栈是 Struts 2 中一个重要的概念，Struts 2 中几乎所有的操作都要与值栈打交道。那么到底什么是值栈呢？值栈是 Struts 2 框架的核心组件，它提供对上下文信息和执行环境中元素的访问机制。实际上，在 Struts 2 中值栈就是一个存放对象的堆栈，对象以 map 形式存储在这个堆栈中，并且可以通过表达式语言获得堆栈中对象属性的值。

值栈贯穿整个 Action 的生命周期，每个 Action 类的对象实例会拥有一个值栈。当 Struts 2 接收到一个请求后，会先建立 Action 类的对象实例，但并不会调用 Action 方法，而是先将 Action 类的相应属性放到值栈的顶层节点。只是所有的属性值都是默认的值，如 String 类型的属性值为 null，int 类型的属性值为 0 等。

当程序执行时，根据 Action 当前所处的状态，修改值栈中的值。

例如值栈中有 fieldErrors 对象，用于存储错误提示信息。在上面的程序中，fieldErrors 对象初始为空，当执行 this.addFieldError("name","name can not be null")；语句后，fieldErrors 中被放入一个 name 对象，其值为"name can not be null"。

那么还有哪些对象放在值栈中的呢？

1．值栈的存储结构

值栈中主要包括以下几个对象。

(1) 临时对象(Temporay Object)：这些对象在请求处理过程中，由容器自动创建并需要临时保存到值栈中。该对象的值可以随着应用的不同而发生变化，当应用结束时，该对象会被清空。比如在页面中利用 Struts 2 标签输出集合中当前正在迭代的元素的值时，这些值都将以临时对象形式存放到值栈中。

(2) 模型对象(Model Object)：当 Action 实现了 ModelDriven 接口时，模型对象就会被存放在栈中被执行的 Action 前面；否则不存在这个级别的内容。

(3) Action 对象(Action Object)：当每个 Action 请求到来时，容器都会创建一个此 Action 的对象，并存入值栈中，也就是当前正在执行的 Action。这个 Action 对象还携带所有与 Action 执行过程有关的信息。

注意：fieldErrors 对象就是与 Action 执行过程中有关的信息。

(4) 命名对象(Named Object)：任何对象都可以被赋予一个标志符而成为命名对象。主要是 Servlet 作用范围内相关对象的信息，比如 session、request 和 application 等。

2. 值栈中对象的遍历顺序

值栈中的几种对象，以图 13.4 所示的顺序存储。当要在值栈中查找某个值时，会按照从上到下的顺序依次遍历每一个对象，直到找到需要的信息为止。使用值栈最大的好处就是只需知道查找信息的 name 标识，就可以在值栈中进行查找。

栈顶
值栈
临时对象
模型对象
Action对象
命名对象
遍历顺序

图 13.4　对象在值栈中的顺序

3. 值栈的使用方式

(1) 使用<s:debug>标签查看值栈中的值

值栈中的数据，可以在 JSP 页中通过插入<s:debug>标签查看。

例如，上面程序中，在登录页面中插入<s:debug>标签，代码如下：

```
<form action="user/User!login" method="get">
    用户名：<input type="text" name="user.name"/><br>
    密　码：<input type="password" name="user.password"/><br>
    <input type="submit" value="登录">
    <s:fielderror fieldName="name" theme="simple"/>
    <s:fielderror fieldName="password" theme="simple"/>
    <s:property value="errors.name[0]"/>
    <s:debug></s:debug>
</form>
```

当应用部署到 Tomcat 中时，登录页面如图 13.5 所示。



图 13.5　登录页面

在<s:debug>标签处显示超链接，单击该超链接，则显示值栈中存储的信息，如图 13.6 所示。



图 13.6 值栈

在图 13.6 中可以看到，Value Stack Contents 中只有一个对象 com.opensymphony.xwork2.DefaultTextProvider，此对象中只有一个属性 texts，其值为 null。

注意： Value Stack 译成中文也是值栈，它指的是狭义值栈，而本节所讲的值栈指的是广义值栈。在本书以后的章节中，所有的值栈均指广义值栈，用 Value Stack 表示狭义值栈。

当用户未输入用户名并提交时，Action 对象被创建，并将错误提示信息放到 fieldErrors 中，如图 13.7 所示。

可以看到，值栈中压入了 UserAction 对象，其 fieldErrors 属性放入了名为 name 的字段，name 的值是包含一个字符串"name can not be null"的数组。

UserAction 对象还包含 user 属性，它是 User 类的对象，是用于接收参数的域模型。

(2) 在页面输出值栈中的值

在页面中可以使用 Struts 2 标签<s:fielderror>输出值栈中 fieldErrors 里的值。

例如<s:fielderror fieldName="password" theme="simple"/>，显示 fieldErrors 中字段名为"password"的值。theme="simple"表示显示值时的样式为"simple"，如图 13.3 所示。

在页面中还可以使用 Struts 2 标签<s:property>与 OGNL 表达式结合，输出值栈中的值。

其方法是标签的 value 属性值为计算值栈中值的 OGNL 表达式(OGNL 表达式的内容将在第 18 章中详细讲解)。如 value="errors.name[0]"则查找 errors 对象中 name 字段的值。name 可以看成一个数组，name[0]表示显示数组的第一个元素，即字符串"name

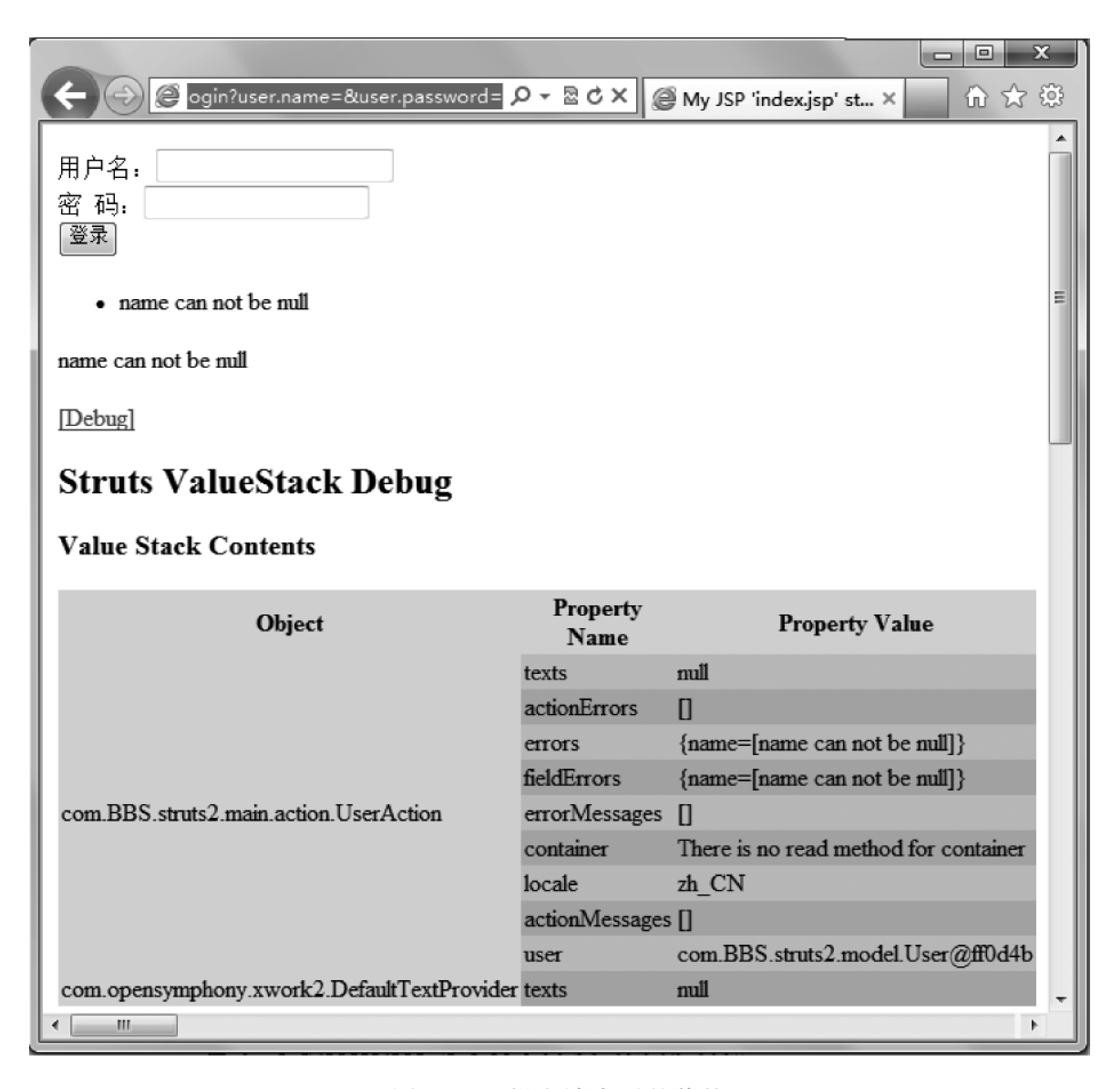


图 13.7　提交请求后的值栈

can not be null"。这样显示的值没有样式设置，界面制作员可以随意设置其样式，如图 13.2 所示。

13.2.3　实践任务 1：Action 类的方法中直接实现输入校验

1. 任务说明

本任务以论坛管理系统登录请求为例，实践在 Action 类的方法中直接添加数据校验代码，实现输入校验。

打开 login.jsp 页面(见图 13.8)，不输入用户名，单击“登录”按钮，提交登录系统的请求。Action 校验数据，返回 login.jsp 页面，显示“name can not be null”提示信息，如图 13.2 所示。

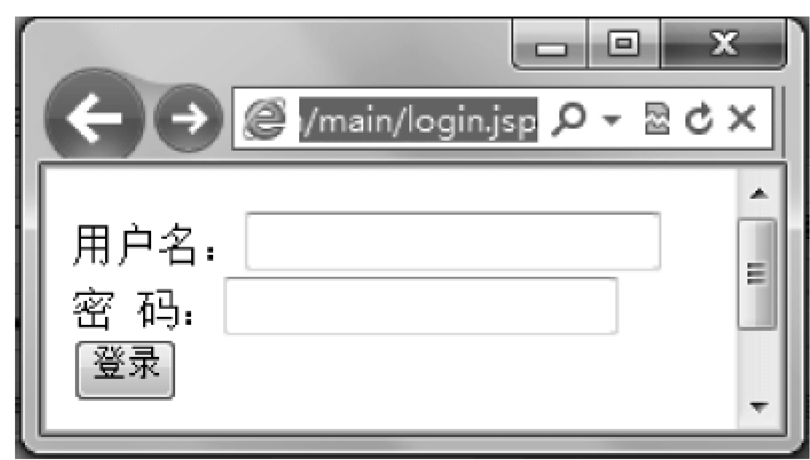


图 13.8　登录页面 login.jsp

打开 login.jsp 页面，输入用户名，不输入密码，单击“登录”按钮，提交登录系统的请求。Action 校验数据，返回 login.jsp 页面，显示“password can

not be null”提示信息，如图 13.3 所示。

2. 任务实施

(1) 创建工程。将工程 BBS_Struts2_0600_ActionDomainModeInput 复制并粘贴，改名为 BBS_Struts2_0800_SimpleDataValiation。

(2) 在 Action 类的 login()方法中插入校验代码。在 Action 类的 login()方法中插入校验代码，并修改其业务逻辑代码，如下所示。

```
package com.BBS.struts2.main.action;

import com.BBS.struts2.model.User;
import com.opensymphony.xwork2.ActionSupport;

public class UserAction extends ActionSupport{

    private User user;

    public String login() {
        //校验代码
        if(user.getName()==null||user.getName().equals("")){
            this.addFieldError("name", "name can not be null");
            return INPUT;
        }
        if(user.getPassword()==null||user.getPassword().equals("")){
            this.addFieldError("password", "password can not be null");
            return INPUT;
        }
        //业务逻辑代码
        if(!user.getName().equals("cvit")){
            this.addFieldError("name", "name is error");
            return INPUT;
        }else if(!user.getPassword().equals("123")){
            this.addFieldError("password", "password is error");
            return INPUT;
        }
        return "success";
    }

    public User getUser() {
        return user;
    }

    public void setUser(User user) {
        this.user = user;
    }
}
```

Action 的 login()方法应该写调用业务逻辑组件的代码，本处为了简单，将业务逻辑处理代码直接写在 login()方法中，在实际开发时，建议不要这么做。

(3) 配置 Action。打开配置文件 struts.xml,输入以下代码。

```
<package name="user" namespace="/user" extends="struts-default">
    <action name="User" class="com.BBS.struts2.main.action.UserAction" >
        <result> /main/index.jsp </result>
    </action>
</package>
<result name="input">/main/login.jsp</result>
```

(4) 编写视图层,提交请求,并设置标签显示错误提示信息。打开 login.jsp,代码如下:

```
<body>
  <form action="user/User!login" method="get">
        用户名: <input type="text" name="user.name"/><br>
        密　码: <input type="password" name="user.password"/><br>
        <input type="submit" value="登录">
        <s:fielderror fieldName="name" theme="simple"/>
        <s:fielderror fieldName="password" theme="simple"/>
        <s:property value="errors.name[0]"/>
        <s:debug></s:debug>
  </form>
</body>
```

(5) 测试。部署工程,在浏览器的地址栏中输入 http://localhost:8080/BBS_Struts2_0800_SimpleDataValiation/main/login.jsp,打开 login.jsp 页面,不输入用户名,单击"登录"按钮,返回 login.jsp 页面,显示提示信息"name can not be null",如图 13.2 所示。

输入用户名,不输入密码,单击"登录"按钮,返回 login.jsp 页面,显示"password can not be null"提示信息,如图 13.3 所示。

试想一下,如果用户名和密码都不输入,会打开哪个 JSP 页面,显示怎样的提示信息。

如果输入的用户名不是"cvit"或密码不是"123",会打开哪个 JSP 页面,并显示怎样的提示信息。

3. 任务总结

本任务通过在 Action 的方法中添加校验代码,实现数据输入校验。

13.2.4　重写 validate()方法实现校验

在上面的例子中,大量的校验代码和业务逻辑代码都放在 login()方法中,降低了程序的可读性,增加了维护的难度。解决办法是将校验代码写到另一个方法——validate()方法中。validate()方法是 ActionSupport 类定义的方法,可以通过重写 validate()方法,将校验代码放在 validate()方法中,从而实现校验代码和业务逻辑代码的分离。

将校验代码写在 validate()方法中,实现与上面例子相同的效果,下面我们一起来看一下具体实现的方法。

(1) 在 Action 的 login()方法中,加入以下校验代码。

```
public class UserAction extends ActionSupport{
  private User user;

  public String login() {
      //业务逻辑代码,处理请求
      ...
      return "success";
  }

  //校验代码
  public void validate(){
      if(user.getName()==null||user.getName().equals("")){
          this.addFieldError("name", "name can not be null");
      }
      if(user.getPassword()==null||user.getPassword().equals("")){
          this.addFieldError("password", "password can not be null");
      }
  }
  ...
}
```

将校验代码写在 validate()方法中,注意 validate()方法的类型是 void,不必写 return 语句。当调用 Action 中任意处理请求的方法(如 login()方法)时,自动先调用 validate()方法。validate()方法执行完毕以后,自动检测 fieldErrors 是否为空,如果不为空,则返回字符串常量 INPUT,login()方法不执行;如果 fieldErrors 为空,则执行 login()方法。

(2) 在 struts.xml 中配置 Action,主要代码如下:

```
<package name="user" namespace="/user" extends="struts-default">
  <action name="User " class="com.BBS.struts2.main.action.UserAction" >
      <result>
          /main/index.jsp
      </result>
      <result name="input">/main/login.jsp</result>
  </action>
</package>
```

(3) 在登录页面显示错误提示信息,主要代码如下:

```
<form action="user/User!login" method="get">
    用户名: <input type="text" name="user.name"/>
    <s:property value="errors.name[0]"/><br>
    密  码: <input type="password" name="user.password"/>
    <s:property value="errors.password[0]"/><br>
    <input type="submit" value="登录">
</form>
```

当用户不输入用户名和密码并提交请求后,自动调用 validate()方法,将错误提示信

息"name can not be null"赋值给 name，并将 name 放到 fieldErrors 中，将错误提示信息"password can not be null"赋值给 password，并将 password 放到 fieldErrors 中，返回字符串常量 INPUT。则界面显示登录页面，并显示 fieldErrors 中的错误提示信息，如图 13.9 所示。

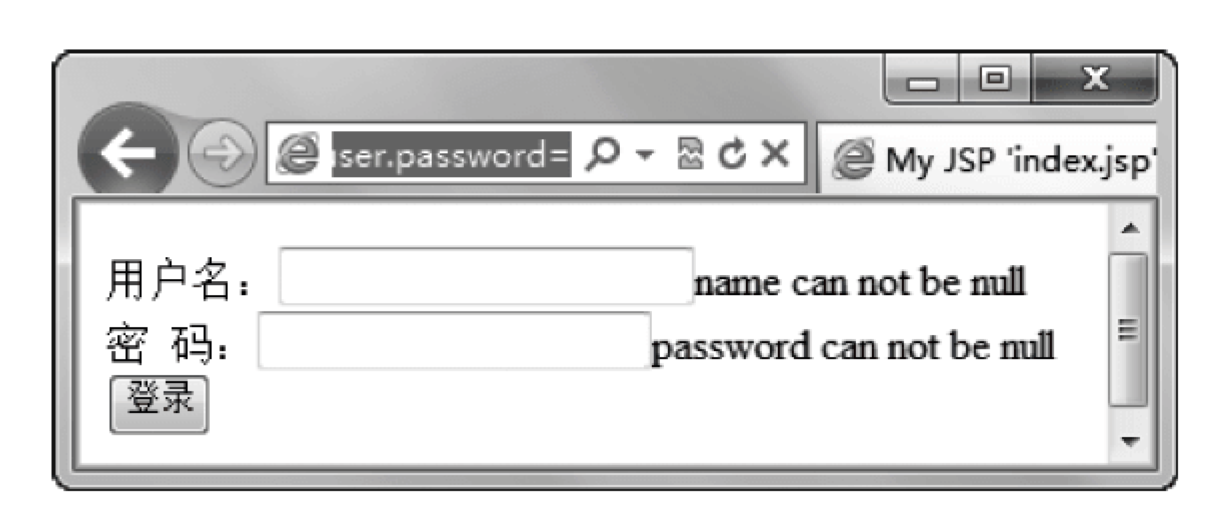


图 13.9　不输入用户名和密码并提交请求后的页面

13.2.5　实践任务 2：重写 validate()方法实现输入校验

1. 任务说明

本任务以论坛管理系统登录请求为例，实践重写 validate()方法，实现输入校验。

打开 login.jsp 页面(见图 13.8)，不输入用户名和密码，单击“登录”按钮，提交登录系统的请求。Action 校验数据，返回 login.jsp 页面，显示“name can not be null”和“password can not be null”提示信息，如图 13.9 所示。

2. 任务实施

(1) 创建工程。将工程 BBS_Struts2_0800_SimpleDataValiation 复制并粘贴，改名为 BBS_Struts2_0900_validate。

(2) 在 Action 类中创建 validate()方法。代码如下：

```
package com.BBS.struts2.main.action;

import com.BBS.struts2.model.User;
import com.opensymphony.xwork2.ActionSupport;

public class User_Action extends ActionSupport{

    private User user;
    //登录方法
    public String login() {
        if(!user.getName().equals("cvit")){
            this.addFieldError("name", "name is error");
            return ERROR;
        }else if(!user.getPassword().equals("123")){
            this.addFieldError("password", "password is error");
            return ERROR;
        }
        return "success";
```

```
    }
    //校验方法
    public void validate() {
        if(user.getName()==null||user.getName().equals("")){
            this.addFieldError("name", "name can not be null");
        }
        if(user.getPassword()==null||user.getPassword().equals("")){
            this.addFieldError("password", "password can not be null");
        }
    }

    public User getUser() {
        return user;
    }

    public void setUser(User user) {
        this.user = user;
    }
}
```

Action 的 login()方法应该写调用业务逻辑组件的代码，本处为了简单，将业务逻辑代码直接写在 login()方法中，在实际开发时，建议不要这么做。

(3) 配置 Action。打开配置文件 struts.xml，输入以下代码。

```
<package name="user" namespace="/user" extends="struts-default">
    <action name="User_login" class="com.BBS.struts2.main.action.User_Action"
            method="login">
        <result>/main/index.jsp</result>
        <result name="error">/main/login_error.jsp</result>
        <result name="input">/main/login.jsp</result>
    </action>
</package>
```

(4) 编写视图层，提交请求，并设置标签显示错误提示信息。打开 login.jsp，代码如下：

```
<body>
    <form action="user/User_login" method="get">
        用户名：<input type="text" name="user.name"/>
        <s:property value="errors.name[0]"/><br>
        密  码：<input type="password" name="user.password"/>
        <s:property value="errors.password[0]"/><br>
        <input type="submit" value="登录">
    </form>
</body>
```

(5) 测试。部署工程，在浏览器的地址栏中输入 http://localhost:8080/BBS_

Struts2_0900_validate/main/login.jsp，打开 login.jsp 页面，不输入用户名和密码，单击“登录”按钮，返回 login.jsp 页面，显示提示信息显示“name can not be null”和“password can not be null”提示信息，如图 13.9 所示。

试想一下，如果输入的用户名不是“cvit”或密码不是“123”，会打开哪个 JSP 页面？并显示怎样的提示信息？

3. 任务总结

本任务通过重写 validate()方法，实现数据输入校验。

13.2.6　validateXxx()方法的使用

我们知道，在一个 Action 中通常需要定义多个方法来处理不同的请求，当我们想对不同的方法指定各自校验逻辑时，validate()方法就不适用了，因为它会对所有的方法都执行输入校验。如果想为不同的方法配置独立的校验规则，可以使用 validateXxx()方法。validateXxx()方法中的 Xxx 代表与之匹配的方法名称，注意方法名的首字母要大写。当然公共的校验逻辑仍可以写在 validate()方法中。

我们以登录和添加用户为例，在 Action 中定义 login()方法和 add()方法，并为其匹配不同的校验逻辑。

(1) Action 类代码如下：

```
package com.BBS.struts2.main.action;
import com.BBS.struts2.model.User;
import com.opensymphony.xwork2.ActionSupport;

public class User_Action extends ActionSupport{

    private User user;
    private String passwordAgain;
    //登录方法
    public String login() {

        System.out.println("------login------");
        if(!user.getName().equals("cvit")){
            this.addFieldError("name", "name is error");
            return ERROR;
        }else if(!user.getPassword().equals("123")){
            this.addFieldError("password", "password is error");
            return ERROR;
        }
        return "success";
    }
    //添加方法
    public String add() {
        System.out.println("------add------");
        return "success";
```

```
    }
    //登录方法的校验逻辑
    public void validateLogin() {
        System.out.println("------validateLogin------");
        if(user.getName()==null||user.getName().equals("")){
            this.addFieldError("name", "name can not be null");
        }
        if(user.getPassword()==null||user.getPassword().equals("")){
            this.addFieldError("password", "password can not be null");
        }
    }
    //添加方法的校验逻辑
    public void validateAdd() {
        System.out.println("------validateAdd------");
        if(user.getName()==null||user.getName().equals("")){
            this.addFieldError("name", "name can not be null");
        }
        if(user.getPassword()==null||user.getPassword().equals("")){
            this.addFieldError("password", "password can not be null");
        }
        if(passwordAgain==null||passwordAgain.equals("")){
            this.addFieldError("passwordAgain", "passwordAgain can not be null");
        }
    ...
}
```

匹配 login()方法的校验方式为 validateLogin()，匹配 add()方法的校验方式为 validateAdd()。注意方法名的首字母是大写的。当 login()方法被调用前，validateLogin()先被调用；当 add()方法被调用前，validateAdd()先被调用。

(2) 校验方法被调用后，自动检测 fieldErrors 是否为空，如果不为空，则返回字符串常量 INPUT。因此在 struts.xml 中配置 Action，主要代码如下：

```
<package name="user" namespace="/user" extends="struts-default">
    <action name="User_login" class="com.BBS.struts2.main.action.User_Action"
            method="login">
        <result>/main/index.jsp </result>
        <result name="input">/main/login.jsp</result>
    </action>

    <action name="User_add" class="com.BBS.struts2.main.action.User_Action"
            method="add">
        <result>/main/User_add_success.jsp </result>
        <result name="input">/main/User_add_input.jsp</result>
    </action>
</package>
```

(3) 设置视图层。

在登录页面显示错误提示信息，主要代码如下：

```
<form action="user/User_login" method="get">
    用户名：<input type="text" name="user.name"/>
    <s:property value="errors.name[0]"/><br>
    密　码：<input type="password" name="user.password"/>
    <s:property value="errors.password[0]"/><br>
    <input type="submit" value="登录">
</form>
```

在添加用户页面显示错误提示信息，主要代码如下：

```
<body>
  新建用户
   <form action="user/User_add" method="get">
       用户名：<input type="text" name="user.name"/>
       <s:property value="errors.name[0]"/><br>
       密　码：<input type="password" name="user.password"/>
       <s:property value="errors.password[0]"/><br>
       确认密 码：<input type="password" name="passwordAgain"/>
       <s:property value="errors.passwordAgain[0]"/><br>
       <input type="submit" value="添加">
   </form>
</body>
```

当用户未输入用户名和密码，提交登录请求时，系统调用 validateLogin()方法，界面显示如图 13.9 所示。

当用户未输入用户名、密码和确认密码，并提交添加用户请求时，系统调用 validateAdd()方法，界面显示如图 13.10 所示。

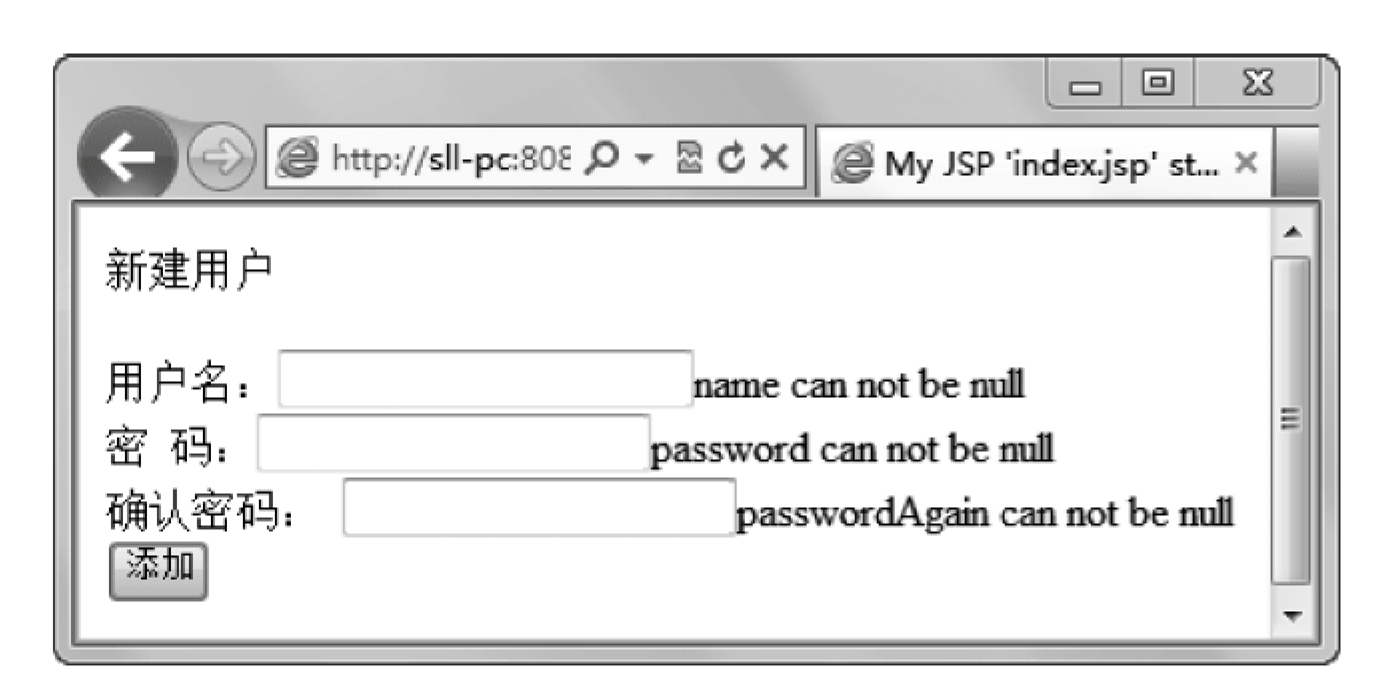


图 13.10　未输入用户名、密码和确认密码并提交请求后的页面

13.2.7　实践任务 3：使用 validateXxx()方法实现输入校验

1. 任务说明

本任务以论坛管理系统登录和添加用户为例，实践使用 validateXxx()方法，实现输入校验。

打开 login.jsp 页面(见图 13.8)，不输入用户名和密码，单击“登录”按钮，提交登录系

统的请求；Action 校验数据，返回 login.jsp 页面，显示“name can not be null”和“password can not be null”提示信息，如图 13.9 所示。

打开 User_add_input.jsp 页面（见图 13.11），不输入用户名、密码和确认密码，单击“添加”按钮，提交添加用户的请求；Action 校验数据，返回 User_add_input.jsp 页面，显示“name can not be null”、“password can not be null”和“passwordAgain can not be null”提示信息，如图 13.10 所示。

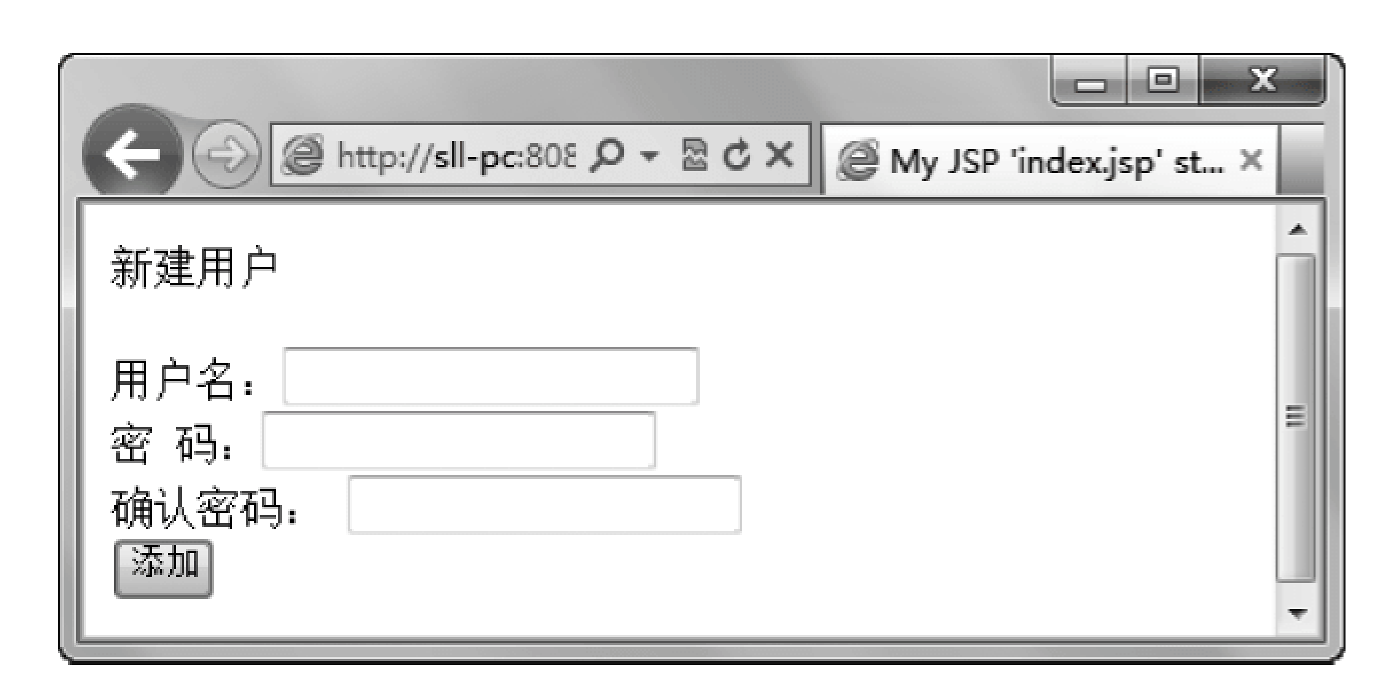


图 13.11　添加用户页面

2. 任务实施

（1）创建工程。将工程 BBS_Struts2_0900_validate 复制并粘贴，改名为 BBS_Struts2_1000_validateMethod。

（2）在 Action 类中创建 add()方法、validateLogin()方法和 validateAdd()方法。代码如下：

```
package com.BBS.struts2.main.action;

import com.BBS.struts2.model.User;
import com.opensymphony.xwork2.ActionSupport;

public class User_Action extends ActionSupport{

    private User user;
    private String passwordAgain;

    public String login() {
        System.out.println("------login------");
        if(!user.getName().equals("cvit")){
            this.addFieldError("name", "name is error");
            return ERROR;
        }else if(!user.getPassword().equals("123")){
            this.addFieldError("password", "password is error");
            return ERROR;
        }
        return "success";
    }

    public String add() {
```

```
        System.out.println("------add------");
        return "success";
    }

    public void validateLogin() {
        System.out.println("------validateLogin------");
        if(user.getName()==null||user.getName().equals("")){
            this.addFieldError("name", "name can not be null");
        }
        if(user.getPassword()==null||user.getPassword().equals("")){
            this.addFieldError("password", "password can not be null");
        }
    }

    public void validateAdd() {
        System.out.println("------validateAdd------");
        if(user.getName()==null||user.getName().equals("")){
            this.addFieldError("name", "name can not be null");
        }
        if(user.getPassword()==null||user.getPassword().equals("")){
            this.addFieldError("password", "password can not be null");
        }
        if(passwordAgain==null||passwordAgain.equals("")){
            this.addFieldError("password", "passwordAgain can not be null");
        }

        if(passwordAgain.equals(user.getPassword())){
            this.addFieldError("passwordAgain","passwordAgain not equals password");
        }
    }

    public User getUser() {
        return user;
    }
    public void setUser(User user) {
        this.user = user;
    }
    public String getPasswordAgain() {
        return passwordAgain;
    }
    public void setPasswordAgain(String passwordAgain) {
        this.passwordAgain = passwordAgain;
    }
}
```

(3) 配置 Action。打开配置文件 struts.xml，输入以下代码。

```
<package name="user" namespace="/user" extends="struts-default">
    <action name="User_login" class="com.BBS.struts2.main.action.User_Action"
            method="login">
        <result>/main/index.jsp </result>
        <result name="input">/main/login.jsp</result>
    </action>
```

```
<action name="User_add" class="com.BBS.struts2.main.action.User_Action"
        method="add">
    <result>/main/User_add_success.jsp </result>
    <result name="input">/main/User_add_input.jsp</result>
</action>
</package>
```

(4) 编写视图层，提交请求，并设置标签显示错误提示信息。打开 login.jsp，代码如下：

```
<body>
  <form action="user/User_login" method="get">
      用户名：<input type="text" name="user.name"/>
      <s:property value="errors.name[0]"/><br>
      密  码：<input type="password" name="user.password"/>
      <s:property value="errors.password[0]"/><br>
      <input type="submit" value="登录">
  </form>
</body>
```

创建 User_add_input.jsp，代码如下：

```
<body>
 新建用户
  <form action="user/User_add" method="get">
      用户名：<input type="text" name="user.name"/>
      <s:property value="errors.name[0]"/><br>
      密  码：<input type="password" name="user.password"/>
      <s:property value="errors.password[0]"/><br>
      确认密码：<input type="password" name="passwordAgain"/>
      <s:property value="errors.passwordAgain[0]"/><br>
      <input type="submit" value="添加">
  </form>
</body>
```

创建 User_add_success.jsp，代码如下：

```
<body>
    添加用户成功 <br>
</body>
```

(5) 测试。部署工程，在浏览器的地址栏中输入 http://localhost:8080/BBS_Struts2_1000_validateMethod/main/login.jsp，打开 login.jsp 页面，不输入用户名和密码，单击“登录”按钮，返回 login.jsp 页面，显示提示信息“name can not be null”和“password can not be null”，如图 13.9 所示。

在浏览器的地址栏中输入 http://localhost:8080/BBS_Struts2_1000_validateMethod/main/User_add_input.jsp，打开 User_add_input.jsp 页面，不输入用户名、密码和确认密码，单击“添加”按钮，返回 User_add_input.jsp 页面，显示提示信息“name can not be null”、“password can not be null”和“passwordAgain can not be null”，如图 13.10 所示。

3. 任务总结

本任务通过使用 validateXxx()方法,实现数据输入校验。

13.2.8　拓展任务:实现后台用户管理模块中数据输入校验

1. 任务要求

实现登录、添加用户、修改用户的输入校验,并将三个输入校验所共有的校验逻辑写在 validate()方法中,不同的校验逻辑写在 validateXxx()方法。

2. 建议步骤

(1) 打开工程——BBS_Struts2_1000_validateMethod。

(2) 创建视图层文件 User_modify_input.jsp 和 User_modify_success.jsp,并编写页面提示信息等代码。

(3) 改写 UserAction 类,创建 modify()方法。

(4) 改写 Action 类,重写 validate()方法和创建 validateModify()方法,并改写 validateLogin()方法和 validateAdd()方法。

(5) 编写 struts.xml 文件,配置 Action。

(6) 打开 User_modify_input.jsp 文件,编写修改用户的请求和插入错误提示信息代码。

(7) 部署工程。

(8) 调试系统。打开浏览器,输入客户端请求,测试程序,如有错误,进行修改。

13.3　常用内置校验器

通过手动在 Action 中编写校验代码,可以完成输入校验工作。但是在企业级应用系统中,会需要大量的输入校验,如果全部以手动编程方法实现的话,工程将会臃肿不堪,难以维护和管理。

Struts 2 的校验框架可以为我们解决这个问题。

Struts 2 将一些常用的校验功能进行了封装,作为其内置的校验器。当需要实现校验功能时,将内置的校验器在 xml 配置文件中用少量的配置代码进行配置即可。

13.3.1　使用 Struts 校验框架完成输入校验的步骤

要使用 Struts 框架完成输入校验,可以通过以下两步完成。

(1) 创建并编辑校验配置文件。

其命名规则为 *-validatin.xml。要对哪个 Action 进行输入校验,* 号处写 Action 的类名。例如对 UserAction 进行输入校验,配置文件名为 UserAction-validatin.xml。

如果一个 Action 对应多个逻辑处理方法,例如:

```
<action name="User_add" class="com.BBS.struts2.main.action.UserAction"
        method="add">
    <result>/main/User_add_success.jsp</result>
</action>
<action name="login" class="com.BBS.struts2.main.action.UserAction" method="login">
    <result>/main/User_list_success.jsp</result>
</action>
```

使用指定校验某个特定方法的方式，其命名规则为 * - * -validatin. xml，第一个 * 号片写 Action 类名，第二个 * 号处写在 struts 配置文件中<action>元素的 name 属性的值。

如 UserAction-User_add-validatin. xml 对 UserAction 的 add 方法输入的数据进行校验，如 UserAction-login-validatin. xml 对 UserAction 的 login 方法输入的数据进行校验。

(2) 保证配置文件与它对应的 Action 类文件在同一目录下。

当调用 Action 中处理请求的方法时，Struts 2 框架会到其目录下查看是否有校验配置文件，如果有，则根据校验配置文件中的校验规则进入数据，输入校验，当数据不符合校验规则时，错误提示信息会被自动放在 fieldErrors 中，返回逻辑视图 INPUT。

13.3.2 注册校验器

Struts 2 的常用内置校验器已配置在 default. xml 文件中。default. xml 在 xwork-core-2.2.1.jar 中，路径为 com/opensymphony/xwork2/validator/validators，它是 Struts 2 默认的校验器注册文件，该文件的主要代码如下：

```
<validators>
<!--必填校验器 -->
<validator name="required"
        class="com.opensymphony.xwork2.validator.validators.RequiredFieldValidator"/>
<!--必填字符串校验器-->
<validator name="requiredstring"
        class="com.opensymphony.xwork2.validator.validators.RequiredStringValidator"/>
<!-- 整数校验器 -->
<validator name="int" class=
        "com.opensymphony.xwork2.validator.validators.IntRangeFieldValidator"/>
<!-- -->
<validator name="long"
        class="com.opensymphony.xwork2.validator.validators.LongRangeFieldValidator"/>
<validator name="short"
        class="com.opensymphony.xwork2.validator.validators.ShortRangeFieldValidator"/>
<validator name="double"
        class="com.opensymphony.xwork2.validator.validators.DoubleRangeFieldValidator"/>
<!-- 日期校验器 -->
<validator name="date"
        class="com.opensymphony.xwork2.validator.validators.DateRangeFieldValidator"/>
```

```
<!-- 表达式校验器 -->
<validator name="expression"
        class="com.opensymphony.xwork2.validator.validators.ExpressionValidator"/>
<!-- 字段表达式校验器 -->
<validator name="fieldexpression"
        class="com.opensymphony.xwork2.validator.validators.FieldExpressionValidator"/>
<!-- 邮件校验器 -->
<validator name="email"
        class="com.opensymphony.xwork2.validator.validators.EmailValidator"/>
<!-- 网址校验器 -->
<validator name="url"
        class="com.opensymphony.xwork2.validator.validators.URLValidator"/>
<validator name="visitor"
        class="com.opensymphony.xwork2.validator.validators.VisitorFieldValidator"/>
<!-- 转换器校验器 -->
<validator name="conversion"
        class="com.opensymphony.xwork2.validator.validators.ConversionErrorFieldValidator"/>
<!-- 字符串长度校验器 -->
<validator name="stringlength"
        class="com.opensymphony.xwork2.validator.validators.StringLengthFieldValidator"/>
<!-- 正则表达式校验器 -->
<validator name="regex"
        class="com.opensymphony.xwork2.validator.validators.RegexFieldValidator"/>
<validator name="conditionalvisitor"
        class="com.opensymphony.xwork2.validator.validators.ConditionalVisitorFieldValidator"/>
</validators>
```

上面代码中注册的校验器，就是 Struts 2 的内建校验器。当进行校验工作时，default.xml 文件会被自动加载到系统中。

提示：通过上面的代码，我们可以知道注册一个校验的规则：使用<validator>元素，可以注册一个校验器，<validator>元素的 name 属性指定校验器名字，class 属性指定该校验器的实现类。

然后将创建 validators.xml 文件，将注册代码写在文件中即可。需要注意的是：开发人员自己创建的 validators.xml 文件要写在工程目录的 WEB-INF/classes 路径下。且当开发人员自己创建 validators.xml 文件后，Struts 2 框架的原有的 default.xml 文件将不会被自动加载，因此需要将原 default.xml 文件中的内容复制到 validators.xml 文件中。

下面一一解释内置校验器的含义和用法。

13.3.3　校验器的配置风格

Struts 2 提供字段校验器风格和非字段校验器风格两种方式来配置校验规则。两者没有本质区别，只是组织校验的规则方式不同：一种是校验器优先，称为非字段校验器风格；另一种是字段优先，称为字段校验器风格。

13.3.4 必填校验器

必填校验器的名字是 required,该校验器要求指定的字段必须有值(非空),该校验器可以接收如下一个参数。

fieldName:该参数指定校验的 Action 属性名,如果采用字段校验器风格,则无须指定该参数。

采用非字段校验器配置风格时,该校验器的配置示例如下:

```
<validators>
  <!-- 使用非字段校验器风格来配置必填校验器 -->
  <field name="username">
    <field-validator type="required">
      <!-- 指定校验失败的提示信息 -->
      <message>username can not be null</message>
    </field-validator>
  …
  </field>
…
</validators>
```

采用字段校验器配置风格时,该校验器的配置示例如下:

```
<validators>
    <!-- 使用字段校验器风格来配置必填校验器,校验 username 属性 -->
    <field name="username">
        <field-validator type="required">
            <!-- 指定校验失败的提示信息 -->
            <message>username can not be null</message>
        </field-validator>
        …
    </field>
    …
</validators>
```

13.3.5 必填字符串校验器

必填字符串校验器的名字是 requiredstring,该校验器要求字段值必须非空且长度大于 0,即该字符串不能是“”。该校验器可以接收如下参数。

(1) fieldName:该参数指定校验的 Action 属性名,如果采用字段校验器风格,则不必指定该参数。

(2) trim:是否在校验前截断被校验属性值前后的空白,该属性是可选的,默认是 true。

采用非字段校验器配置风格时,该校验器的配置示例如下:

```
<validators>
  <!-- 使用非字段校验器风格来配置必填字符串校验器 -->
  <validator type="requiredstring">
    <!-- 指定需要校验的字段名 -->
    <param name="fieldName">username</param>
    <!-- 指定去掉被校验属性值前后的空白 -->
    <param name="trim">true</param>
    <!-- 指定校验失败的提示信息 -->
    <message>username is requiredString</message>
  </validator>
…
</validators>
```

采用字段校验器配置风格时，该校验器的配置示例如下：

```
<validators>
  <!-- 使用字段校验器风格来配置必填字符串校验器，校验 username 属性 -->
  <field name="username">
    <field-validator type="requiredstring">
    <!-- 指定去掉被校验属性值前后的空白 -->
      <param name="trim">true</param>
      <!-- 指定校验失败的提示信息 -->
      <message>username is requiredString </message>
    </field-validator>
  …
  </field>
…
</validators>
```

13.3.6　实践任务 4：使用 Struts 校验框架实现必填字符串校验

1. 任务说明

本任务以论坛管理系统登录为例，实践使用 Struts 校验框架，实现必填字符串校验。

打开 login.jsp 页面(见图 13.8)，不输入用户名和密码，单击“登录”按钮，提交登录系统的请求；Struts 校验框架校验输入数据，返回 login.jsp 页面，显示“请输入用户名”和“请输入密码”提示信息，如图 13.12 所示。



图 13.12　未填写用户和密码的提示信息

2. 任务实施

(1) 创建工程。创建 Struts 2 应用工程 BBS_Struts2_1100_validateFrame。

(2) 视图层文件。创建 login.jsp,主要代码如下:

```
<form action="" method="get">
    用户名: <input type="text" name="name"/><br>
    密  码: <input type="password" name="password"/><br>
    <input type="submit" value="登录">
</form>
```

创建 index.jsp,主要代码如下:

```
<body>
    后台主页页面 <br>
</body>
```

(3) 创建 Action 类。创建 LoginAction,代码如下:

```
package com.BBS.struts2.main.action;

import com.opensymphony.xwork2.ActionSupport;

public class LoginAction extends ActionSupport{

    private String name;
    private String password;

    public String login() {
        return SUCCESS;
    }
    //属性的 getter 和 setter 方法略
    ...
}
```

(4) 创建校验配置文件。创建 LoginAction-validation.xml,代码如下:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE validators PUBLIC "-//OpenSymphony Group//XWork Validator 1.0.2//EN"
"http://www.opensymphony.com/xwork/xwork-validator-1.0.2.dtd">
<validators>
    <field name="name">
        <field-validator type="requiredstring">
            <message>请输入用户名</message>
        </field-validator>
    </field>

    <field name="password">
        <field-validator type="requiredstring">
            <message>请输入密码</message>
        </field-validator>
```

```
        </field>
</validators>
```

将 LoginAction-validation.xml 放在与 LoginAction.java 相同的目录下。

(5) 配置 Action。打开配置文件 struts.xml，输入以下代码。

```
<package name="user" namespace="/user" extends="struts-default">
  <action name="User_login" class="com.BBS.struts2.main.action.User_Action"
          method="login">
    <result>/main/index.jsp </result>
    <result name="input">/main/login.jsp</result>
  </action>
</package>
```

(6) 编写视图层，提交请求，并设置标签显示错误提示信息。打开 login.jsp，代码如下：

```
<body>
   <form action="user/User_login" method="get">
       用户名：<input type="text" name="user.name"/>
       <s:property value="errors.name[0]"/><br>
       密　码：<input type="password" name="user.password"/>
       <s:property value="errors.password[0]"/><br>
       <input type="submit" value="登录">
       <s:debug></ s:debug>
   </form>
</body>
```

fieldErrors 中的内容会被自动放在 errors 中，因此使用<s:property value="errors.password[0]"/>同样可以输出 password 中存放的错误提示信息。

(7) 测试。部署工程，在浏览器的地址栏中输入 http://localhost:8080/BBS_Struts2_1100_validateFrame/main/login.jsp，打开 login.jsp 页面，不输入用户名和密码，单击"登录"按钮，返回 login.jsp 页面，显示提示信息"请输入用户名"和"请输入密码"，如图 13.9 所示。

单击 s:debug 超链接，查看 fieldErrors 中的值，如图 13.13 所示。

3. 任务总结

本任务通过使用 Struts 2 校验框架的必填字符串校验器，实现数据输入校验。

13.3.7　整数校验器

整数校验器包括 int、long、short，该校验器要求字段的整数值必须在指定范围内。该校验器可以接收如下参数。

(1) fieldName：该参数指定校验的 Action 属性名，如果采用字段校验器风格，则不必指定该参数。

(2) min：指定该属性的最小值，该参数可选，如果没有指定，则不检查最小值。



图 13.13　查看 fieldErrors 中的值

(3) max：指定该属性的最大值，该参数可选，如果没有指定，则不检查最大值。

采用非字段校验器配置风格时，该校验器的配置示例如下：

```
<validators>
  <!-- 使用非字段校验器风格来配置整数校验器 -->
  <validator type="int">
    <!-- 指定需要校验的字段名 -->
    <param name="fieldName">age</param>
    <!-- 指定 age 属性的最小值 -->
    <param name="min">24</param>
    <!-- 指定 age 属性的最大值 -->
    <param name="max">60</param>
    <!-- 指定校验失败的提示信息 -->
    <message>Age needs to be between ${min} and ${max}</message>
  </validator>
  ...
```

```
</validators>
```

采用字段校验器配置风格时，该校验器的配置示例如下：

```
<validators>
  <!-- 使用字段校验器风格来配置整数校验器，校验 age 属性 -->
  <field name="age">
    <field-validator type="int">
      <!-- 指定 age 属性的最小值 -->
      <param name="min">24</param>
      <!-- 指定 age 属性的最大值 -->
      <param name="max">60</param>
      <!-- 指定校验失败的提示信息 -->
      <message>Age needs to be between ${min} and ${max}</message>
    </field-validator>
  …
  </field>
  …
</validators>
```

双精度浮点数校验器与整数校验器用法几乎相同，唯一的区别是它要求被校验的 Action 属性是双精度浮点数。

13.3.8　日期校验器

日期校验器的名字是 date，该校验器要求字段的日期值必须在指定范围内。该校验器可以接受如下参数。

(1) fieldName：该参数指定校验的 Action 属性名，如果采用字段校验器风格，则不必指定该参数。

(2) min：指定该属性的最小值，该参数选用，如果没有指定，则不检查最小值。

(3) max：指定该属性的最大值，该参数选用，如果没有指定，则不检查最大值。

注意：*系统默认使用类型转换器将字符串转换为"四位数字表示年-两位数字表示月-两位数字表示日"格式的日期类型的数据。*

采用非字段校验器配置风格时，该校验器的配置示例如下：

```
<validators>
  <!-- 使用非字段校验器风格来配置日期校验器 -->
  <validator type="date">
    <!-- 指定需要校验的字段名 -->
    <param name="fieldName">birthday</param>
    <!-- 指定 birthday 属性的最小值 -->
    <param name="min">2000-01-01</param>
    <!-- 指定 birthday 属性的最大值 -->
    <param name="max">2012-12-31</param>
    <!-- 指定校验失败的提示信息 -->
    <message>Birthday needs to be between ${min} and ${max}</message>
```

```
    </validator>
    …
</validators>
```

采用字段校验器配置风格时，该校验器的配置示例如下：

```
<validators>
  <!-- 使用字段校验器风格来配置整数校验器，校验 birthday 属性 -->
  <field name="birthday">
    <field-validator type="date">
      <!-- 指定 birthday 属性的最小值 -->
      <param name="min">2000-01-01</param>
      <!-- 指定 birthday 属性的最大值 -->
      <param name="max">2012-12-31</param>
      <!-- 指定校验失败的提示信息 -->
      <message>Birthday needs to be between ${min} and ${max}</message>
    </field-validator>
  …
  </field>
  …
</validators>
```

13.3.9 表达式校验器

表达式校验器的名字是 expression，它是一个非字段校验器，不可在字段校验器的配置风格中使用。该表达式校验器要求 OGNL 表达式返回 true，当返回 true 时，该校验通过；否则，校验没有通过。

该校验器可以接收一个参数。

expression：该参数指定一个逻辑表达式，该逻辑表达式基于 ValueStack 进行求值，最后返回一个 Boolean 值；当返回 true 时，校验通过；否则校验失败。

采用非字段校验器配置风格时，该校验器的配置示例如下：

```
<validators>
  <!-- 使用非字段校验器风格来配置表达式校验器 -->
  <validator type="expression">
    <!-- 指定需要校验表达式-->
    <param name="fieldName">…</param>
    <!-- 指定校验失败的提示信息 -->
    <message>Failed </message>
  </validator>
  …
</validators>
```

注意：OGNL 表达式我们在后面的章节中讲解。

13.3.10 字段表达式校验器

字段表达式校验器的名字是 fieldexpression，它要求指定字段满足一个逻辑表达式。

该校验器可以接收如下两个参数。

(1) fieldName：该参数指定校验的 Action 属性名，如果采用字段校验器风格，则不必指定该参数。

(2) expression：该参数指定一个逻辑表达式，该逻辑表达式基于 ValueStack 进行求值，最后返回一个 Boolean 值；当返回 true 时，校验通过；否则校验失败。

采用非字段校验器配置风格时，该校验器的配置示例如下：

```
<validators>
  <!-- 使用非字段校验器风格来配置表达式校验器 -->
  <validator type="fieldexpression">
    <!-- 指定需要校验的字段名-->
    <param name="fieldName">password</param>
    <!-- 指定需要校验的字段名-->
    <param name="expression"><!CDATA(password==rPassword)]]</param>
    <!-- 指定校验失败的提示信息 -->
    <message>密码和确认密码必须一致</message>
  </validator>
  ...
</validators>
```

password 和 rPassword 的值分别为用户输入的密码和确认密码。

采用字段校验器配置风格时，该校验器的配置示例如下：

```
<validators>
  <!-- 使用字段校验器风格来配置字段表达式校验器，校验 password 属性 -->
  <field name="password">
    <field-validator type="fieldexpression">
      <!-- 指定逻辑表达式-->
      <param name="expression"><!CDATA(password==rPassword)]]</param>
      <!-- 指定校验失败的提示信息 -->
      <message>密码和确认密码必须一致</message>
    </field-validator>
  ...
  </field>
  ...
</validators>
```

13.3.11　邮件地址校验器

邮件地址校验器的名称是 E-mail，它要求被检查字段的字符如果非空，则必须是合法的邮件地址。不过这个校验器其实就是基于正则表达式进行校验的，系统的邮件地址正则表达如下：

```
\\b(^[_A-Za-z0-9-](\\.[_A-Za-z0-9-])*@([A-Za-z0-9-])+((\\.com)|(\\.net)|(\\.org)|
(\\.info)|(\\.edu)|(\\.mil)|(\\.gov)|(\\.biz)|(\\.ws)|(\\.us)|(\\.tv)|(\\.cc)|(\\.aero)|
(\\.arpa)|(\\.coop)|(\\.int)|(\\.jobs)|(\\.museum)|(\\.name)|(\\.pro)|(\\.travel)|
(\\.nato)|(\\..{2,3})|(\\..{2,3}\\..{2,3}))$)\\b
```

注意：随着技术的不断发展，有可能上面的正则表达式不能完全覆盖实际的电子邮件地址。因此，建议开发人员使用正则表达式校验器来完成邮件校验。

该校验器可以接收如下一个参数。

fieldName：该参数指定校验的 Action 属性名，如果采用字段校验器风格，则不必指定该参数。

采用非字段校验器配置风格时，该校验器的配置示例如下：

```
<field-validator type="email">
    <message>电子邮件地址无效</message>
</field-validator>
```

13.3.12 网址校验器

网址校验器的名称是 url，它要求被检查字段的字符如果非空，则必须是合法的 URL 地址。不过这个校验器其实是基于正则表达式进行校验的，因此，有可能随着技术的发展，这个校验器不能完全覆盖所有的网址，也建议开发人员使用正则表达式校验器进行网址校验。

该校验器可以接收如下一个参数。

fieldName：该参数指定校验的 Action 属性名，如果采用字段校验器风格，则无需指定该参数。

采用非字段校验器配置风格时，该校验器的配置示例如下：

```
<field-validator type="url">
    <param name="fieldName">url</param>
    <message>您的网址无效</message>
</field-validator>
```

13.3.13 转换校验器

转换校验器的名称是 conversion，它检查被校验字段在类型转换过程中是否出现错误。它可以接收如下两个参数。

(1) fieldName：该参数指定校验的 Action 属性名，如果采用字段校验器风格，则不必指定该参数。

(2) repopulateField：该参数指定当类型转换失败后，返回 input 页面时，类型转换失败的表单域是否保留原来的错误输入。

采用非字段校验器配置风格时，该校验器的配置示例如下：

```
<validator type="conversion">
    <param name="fieldName">date</param>
    <!-- 指定类型转换失败后，返回输入页面保留原来的错误输入-->
    <param name ="repopulateField">true</param>
    <message>您所输入的日期格式有误</message>
```

```
</validator>
```

采用字段校验器配置风格时，该校验器的配置示例如下：

```
<field name="age">
    <field-validator type="conversion">
        <message>您输入的年龄必须是数字</message>
        <!-- 指定类型转换失败后，返回输入页面不保留原来的错误输入-->
        <param name ="repopulateField">false</param>
    </field-validator>
</field>
```

13.3.14 字符串长度校验器

字符串长度校验器的名称是 stringlength，它要求被校验字段的长度必须在指定的范围之内，否则就算校验失败。该校验器可以接收如下几个参数。

(1) q fieldName：该参数指定校验的 Action 属性名，如果采用字段校验器风格，则不必指定该参数。

(2) q maxLength：该参数指定字段值的最大长度，该参数选用，如果不指定该参数，则最大长度不受限制。

(3) q minLength：该参数指定字段值的最小长度，该参数选用，如果不指定该参数，则最小长度不受限制。

(4) q trim：指定校验该字段之前是否截断该字段值前后的空白。该参数选用，默认是 true。

采用非字段校验器配置风格时，该校验器的配置示例如下：

```
<validator type="stringlength">
    <param name="fieldName">username</param>
    <param name="minLength">4</param>
    <param name="maxLength">10</param>
    <message>用户名长度在${minLength}到${maxLength}之间</message>
</validator>
```

采用字段校验器配置风格时，该校验器的配置示例如下：

```
<field name="username">
    <field-validator type="stringlength">
        <param name="minLength">4</param>
        <param name="maxLength">10</param>
        <param name="trim">true</param>
        <message>用户名长度在${minLength}到${maxLength}之间</message>
    </field-validator>
</field>
```

13.3.15 正则表达式校验器

正则表达式校验器的名称是 regex，它检查被校验字段是否匹配一个正则表达式。

该校验器可以接收如下几个参数。

(1) fieldName：该参数指定校验的 Action 属性名，如果采用字段校验器风格，则不必指定该参数。

(2) expression：该参数是必需的，该参数指定匹配用的正则表达式。

(3) caseSensitive：该参数指明进行正则表达式匹配时，是否区分大小写。该参数是选用的，默认是 true。

采用非字段校验器配置风格时，该校验器的配置示例如下：

```
<validator type="regex">
    <param name="fieldName">username</param>
    <param name="expression"><![CDATA[(\w{6,12})]]></param>
    <message>您的用户名只能包含字母和数字,且长度必须在 6 到 12 之间</message>
</validator>
```

采用字段校验器配置风格时，该校验器的配置示例如下：

```
<field name="username">
    <field-validator type="regex">
        <param name="expression"><![CDATA[(\w{6,12})]]></param>
        <message>
            您的用户名只能包含字母和数字,且长度必须在 6 到 12 之间
        </message>
    </field-validator>
</field>
```

13.3.16 Visitor 校验器

Visitor 校验器主要用于检测 Action 里的复合属性，例如，一个 Action 里定义某个模型类的对象，作为 Action 的属性。这是经常会遇到的，因为 Action 在接收参数时，通过会定义模型类或 DTO 的对象用于传递参数。例如，UserAction 中定义了 User 类的对象作为 Action 的属性，代码如下：

```
public class UserAction extends ActionSupport
{
  //Action 里包含了一个 User 类型的属性
  private User user;
  //user 属性的 setter 和 getter 方法
  public void setUser(User user){
      this.user = user;
  }
  public User getUser() {
      return (this.user);
  }
  ...
}
```

User 类包括 3 个属性及其 getter 和 setter 方法，其主要代码如下：

```
public class User
{
    //User 类里包含的 3 个基本数据类型的属性
    private String name;
    private String password;
    private int age;
    //此处省略了 3 个属性的 setter 和 getter 方法
    …
}
```

要校验上面 UserAction 里的 User 属性，显然不能通过其他校验器完成，因为那些普通校验器都只能校验基本数据类型和字符串类型。此时，为了校验该 User 类型属性里的其他属性，则应该使用 Visitor 校验器。

Visitor 校验器采用非字段校验器配置风格时，该校验器的配置示例如下：

```
<validator type="visitor">
    <param name="fieldName">user</param>
    <param name="context">userContext</param>
    <param name="appendPrefix">true</param>
</validator>
```

Visitor 校验器采用字段校验器配置风格时，该校验器的配置示例如下：

```
<!-- 指定校验 user 字段-->
<field name="user">
    <!--使用 visitor 校验器-->
    <field-validator type="visitor">
        <!-- 指定校验规则文件-->
        <param name="context">userContext</param>
        <!-- 指定校验失败后提示信息是否添加下面前缀-->
        <param name="appendPrefix">true</param>
        <message>您输入的用户信息中：</message>
    </field-validator>
</field>
```

但上面的校验并未指定 User 类中各字段的校验规则，因此还需为 User 类指定校验规则文件。在默认情况下，该校验文件的规则文件名为 User-validation.xml，因为配置 Visitor 校验器时指定了 context 为 userContext，则该校验文件的文件名为 User-userContext-validation.xml。并注意 User-userContext-validation.xml 文件放在与 User.class 的同一文件夹下，该文件主要代码如下：

```
<!-- 校验 User 属性的 name 属性 -->
<field name="name">
  <!-- 指定 name 属性必须满足必填规则 -->
  <field-validator type="requiredstring">
    <param name="trim">true</param>
    <!--指定校验失败后提示信息-->
    <message>用户名必须输入</message>
  </field-validator>
```

```
        <!-- 指定 name 属性必须匹配正则表达式 -->
          <field-validator type="regex">
            <param name="expression"><![CDATA[(\w{6,12})]]></param>
            <!-- 指定校验失败后提示信息-->
            <message>用户名只能包含字母和数字,且长度必须在 6 到 12 之间</message>
          </field-validator>
      </field>
      <!-- 校验 User 属性的 password 属性 -->
      <field name="password">
      <!-- 指定 password 属性必须满足必填规则 -->
        <field-validator type="requiredstring">
          <param name="trim">true</param>
          <!--指定校验失败后提示信息-->
          <message>密码必须输入</message>
        </field-validator>
        <!-- 指定 password 属性必须满足匹配指定的正则表达式 -->
        <field-validator type="regex">
          <param name="expression"><![CDATA[(\w{6,12})]]></param>
            <!-- 指定校验失败后提示信息-->
          <message>密码只能包含字母和数字,且长度必须在 6 到 12 之间</message>
        </field-validator>
      </field>
      <!-- 指定 User 属性的 age 属性必须在指定范围内-->
      <field name="age">
        <field-validator type="int">
          <param name="min">1</param>
          <param name="max">150</param>
            <!-- 指定校验失败后提示信息-->
          <message>年龄必须在 1 到 150 岁之间</message>
        </field-validator>
      </field>
```

其视图层的 jsp 文件主要代码如下：

```
<form action="main/User_add" method="post">
    用户名：<input type="text" name="user.name"/><br>
    密  码：<input type="password" name="user.password"/><br>
    年  龄：<input type="text" name="user.age"/><br>
    <input type="submit" value="添加">
</form>
```

如果用户未输入用户名、密码，年龄输入为 0 时，错误提示信息如图 13.14 所示。

13.3.17 使用内置校验器时的校验顺序

在使用 Struts 2 内置校验器时，其校验的顺序如下。

(1) 所有非字段校验风格的校验器，优先于字段校验风格的校验器。

(2) 所有非字段校验风格的校验器中，排在前面的会先执行。

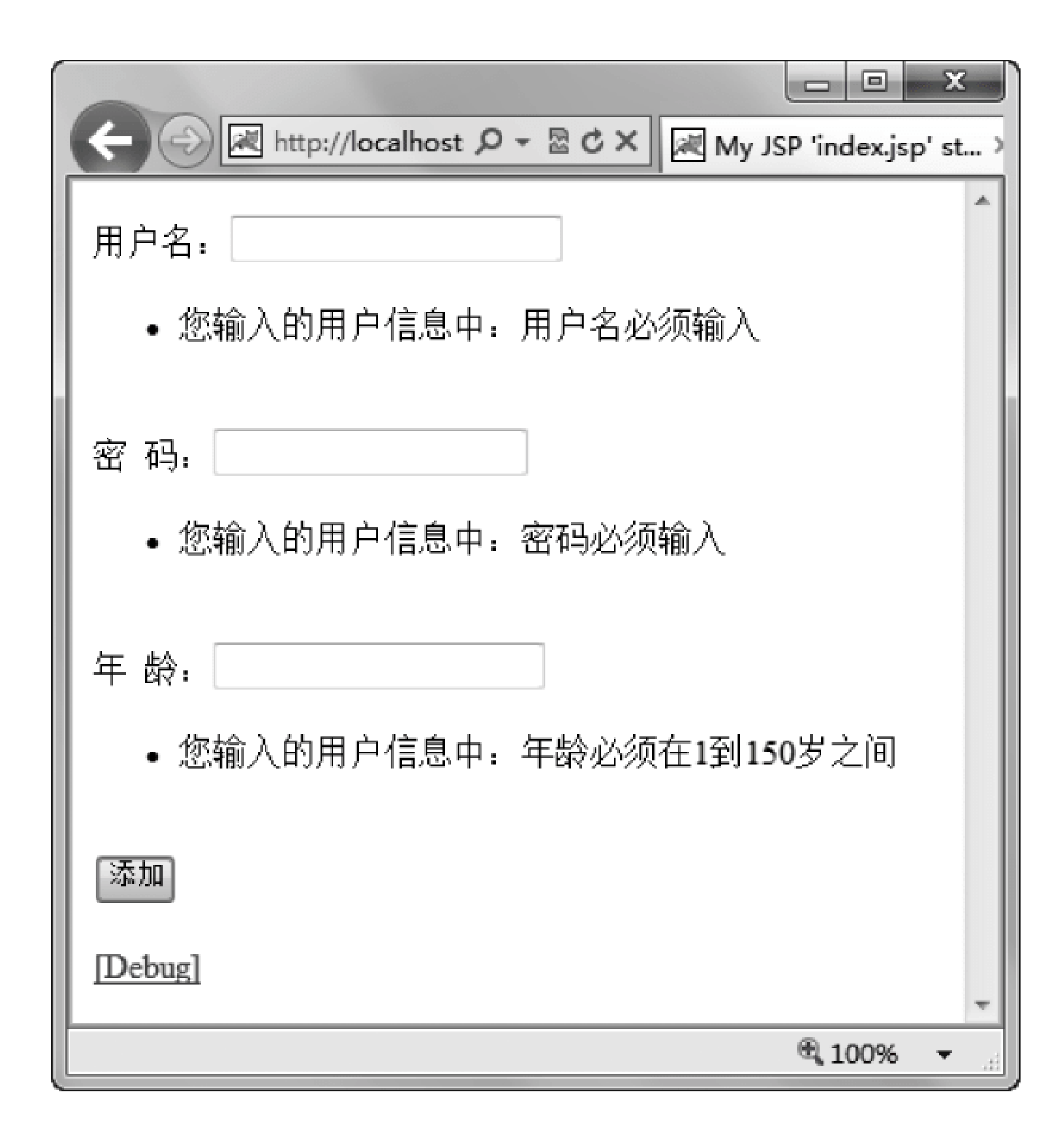


图 13.14 输入校验

(3) 所有字段校验风格的校验器，排在前面的会先执行。

13.3.18 实践任务 5：使用 Struts 校验框架实现数据输入校验

1. 任务说明

本任务以论坛管理系统添加用户为例，实践使用 Struts 校验框架的 Visitor 校验器、整数校验器等常用内置校验器，实现数据输入校验。

打开 User_add_input.jsp 页面，不输入用户名、密码和确认密码，单击“添加”按钮，提交添加用户的请求；Action 校验数据，返回 User_add_input.jsp 页面，显示“用户名必须输入”、“密码必须输入”和“年龄必须在 1 到 150 岁之间”提示信息，如图 13.14 所示。

2. 任务实施

(1) 创建工程。创建 Struts 2 应用工程 BBS_Struts2_1150_validateFrame。

(2) 视图层文件。

① 创建 User_add_input.jsp，主要代码如下：

```
<form action="" method="post">
    用户名：<input type="text" name="user.name"/><br>
    密  码：<input type="password" name="user.password"/><br>
    年  龄：<input type="text" name="user.age"/><br>
    <input type="submit" value="添加">
</form>
```

② 创建 User_add_success.jsp，主要代码如下：

```
<body>
    添加用户成功!<br>
</body>
```

(3) 创建模型类。创建 User 类，代码如下：

```
package com.BBS.struts2.model;

public class User {
    private String name;
    private String password;
    private int age;
    //其属性的 setter 和 getter 方法略
    ...
}
```

(4) 创建 Action 类。创建 UserAction，代码如下：

```
package com.BBS.struts2.main.action;

import com.BBS.struts2.model.User;
import com.opensymphony.xwork2.ActionSupport;

public class UserAction extends ActionSupport{

    private User user;

    public String add() {
        return SUCCESS;
    }

    public User getUser() {
        return user;
    }
    public void setUser(User user) {
        this.user = user;
    }
}
```

(5) 配置 Action。打开配置文件 struts.xml，输入以下代码。

```
<package name="user" namespace="/user" extends="struts-default">
<action name="User_add" class="com.BBS.struts2.main.action.UserAction"
        method="add">
    <result>/main/User_add_success.jsp</result>
    <result name="input">/main/User_add_input.jsp</result>
</action>
</package>
```

(6) 创建校验配置文件。

① 创建 UserAction-User_add-validation.xml,代码如下:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE validators PUBLIC "-//OpenSymphony Group//XWork Validator 1.0.2//EN"
"http://www.opensymphony.com/xwork/xwork-validator-1.0.2.dtd">
<validators>
      <!-- 指定校验 user 字段-->
      <field name="user">
        <!--使用 visitor 校验器-->
        <field-validator type="visitor">
            <!-- 指定校验规则文件-->
            <param name="context">userContext</param>
            <!-- 指定校验失败后提示信息是否添加下面前缀-->
            <param name="appendPrefix">true</param>
            <message>您输入的用户信息中: </message>
        </field-validator>
      </field>
</validators>
```

将 UserAction-User_add-validation.xml 放在与 LoginAction.java 相同的目录下。

其中,UserAction 为 Action 类名,User_add 为 struts.xml 配置文件中,<action>元素 name 属性的值。

② 创建 User-userContext-validation.xml,代码如下:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE validators PUBLIC "-//OpenSymphony Group//XWork Validator 1.0.2//EN"
"http://www.opensymphony.com/xwork/xwork-validator-1.0.2.dtd">
<validators>
    <!-- 校验 User 属性的 name 属性 -->
    <field name="name">
        <!-- 指定 name 属性必须满足必填规则 -->
        <field-validator type="requiredstring">
            <param name="trim">true</param>
            <!--指定校验失败后提示信息-->
            <message>用户名必须输入</message>
        </field-validator>
    <!-- 指定 name 属性必须匹配正则表达式 -->
    <field-validator type="regex">
    <param name="expression"><![CDATA[(\w{6,12})]]></param>
    <!-- 指定校验失败后提示信息-->
    <message>用户名只能包含字母和数字,且长度必须在 6 到 12 之间</message>
    </field-validator>
    </field>
  <!-- 校验 User 属性的 password 属性 -->
  <field name="password">
      <!-- 指定 password 属性必须满足必填规则 -->
      <field-validator type="requiredstring">
          <param name="trim">true</param>
          <!--指定校验失败后提示信息-->
```

```
                <message>密码必须输入</message>
            </field-validator>
            <!-- 指定 password 属性必须满足匹配指定的正则表达式 -->
            <field-validator type="regex">
                <param name="expression"><![CDATA[(\w{6,12})]]></param>
                <!-- 指定校验失败后提示信息-->
                <message>密码只能包含字母和数字,且长度必须在 6 到 12 之间</message>
            </field-validator>
        </field>
        <!-- 校验 User 属性的 age 属性 -->
        <field name="age">
          <!-- 指定 User 属性的 age 属性转换为整数必须成功-->
          <field-validator type="conversion">
          <message>您输入的年龄必须是整数</message>
          <!-- 指定类型转换失败后,返回输入页面不保留原来的错误输入-->
          <param name ="repopulateField">false</param>
          </field-validator>
            <!-- 指定 User 属性的 age 属性必须在指定范围内-->
            <field-validator type="int">
                <param name="min">1</param>
                <param name="max">150</param>
                <!-- 指定校验失败后提示信息-->
                <message>年龄必须在 1 到 150 岁之间</message>
            </field-validator>
        </field>
    </validators>
```

将 User-userContext-validation. xml 放在与 User. java 相同的目录下。

其中,User 为类名,userContext 为 UserAction-User_add-validation. xml 配置文件中,<param name="context">userContext</param>所指定。

(7) 编写视图层,提交请求,并设置标签显示错误提示信息。打开 User_add_input. jsp,代码如下:

```
<body>
   <form action="user/User_add" method="get">
        用户名: <input type="text" name="user.name"/>
        <s:fielderror fieldName="user.name" theme="simple"/><br>
        密　码: <input type="password" name="user.password"/>
        <s:fielderror fieldName="user.password" theme="simple"/><br>
        年　龄: <input type="text" name="user.age"/>
        <s:fielderror fieldName="user.age" theme="simple"/><br>
        <input type="submit" value="添加">
        <s:debug></s:debug>
   </form>
</body>
```

(8) 测试。部署工程,在浏览器的地址栏中输入 http://localhost:8080/BBS_Struts2_1150_validateFrame/main/User_add_input. jsp,打开 User_add_input. jsp 页面,不输入用户名和密码,输入年龄为 0,单击"添加"按钮,返回 User_add_input. jsp 页面,显示提示信息"您输入的用户信息中:用户名必须输入"、"您输入的用户信息中:密码必须

输入”和“您输入的用户信息中：年龄必须在 1 到 150 岁之间”，如图 13.14 所示。

单击<s:debug>标签，打开 ValueStack，查看 fieldErrors 中的值，如图 13.15 所示。

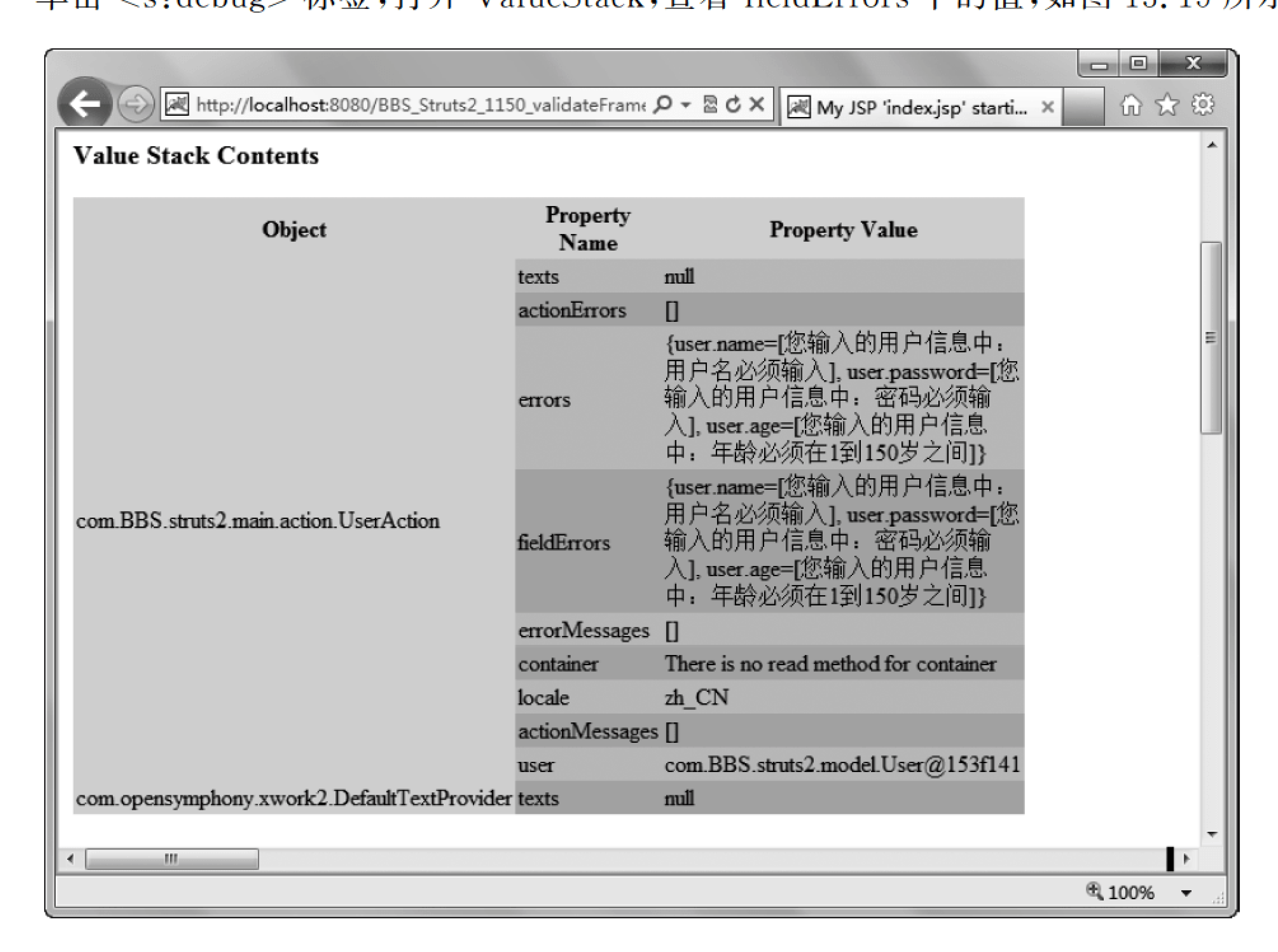


图 13.15　查看 ValueStack

3. 任务总结

本任务通过使用 Struts 2 校验框架的常用内置校验器，实现数据的输入校验。

13.3.19　实训：实现论坛管理系统中添加用户的输入校验

使用常用内置校验器，实现添加用户的输入校验，校验规则如下。

(1) 用户名、密码、确认密码不能为空，只能是包含字母和数字的 4 到 10 位的字符串。

(2) 密码和确认密码必须一致。

(3) 年龄必须为数字，在 20～55 之间。

(4) 生日在 1960-01-01 到 1985-12-31 之间。

13.4　输入校验的流程

(1) 当用户提交带参数(即输入数据)的请求时，类型转换器对请求参数进行类型转换，并把转换后的值设置成 action 的属性值。

(2) 如果在执行类型转换的过程中出现异常，系统会将异常信息保存到 ActionContext，conversionError 拦截器将异常信息封装到 fieldErrors 里，然后执行第(3)

步。如果类型转换没有出现异常,则直接进入第(3)步。

(3) 调用 Struts 2 内置的校验规则进行输入校验(也就是根据各 * validation. xml 文件中定义的校验规则进行输入校验)。

(4) 系统通过反射技术调用 action 中的 validateXxx()方法,其中 Xxx 为 Action 中将处理此请求的处理逻辑所对应的方法名。

(5) 调用 Action 类中的 validate()方法。

(6) 经过上面 5 步,如果系统中的 fieldErrors 存在错误信息,系统自动将请求转发至名称为“input”逻辑视图所指定的视图资历源。如果系统中的 fieldErrors 没有任何错误信息,系统将调用 Action 中的处理用户请求的方法。

图 13.16 展示了上面的处理流程。

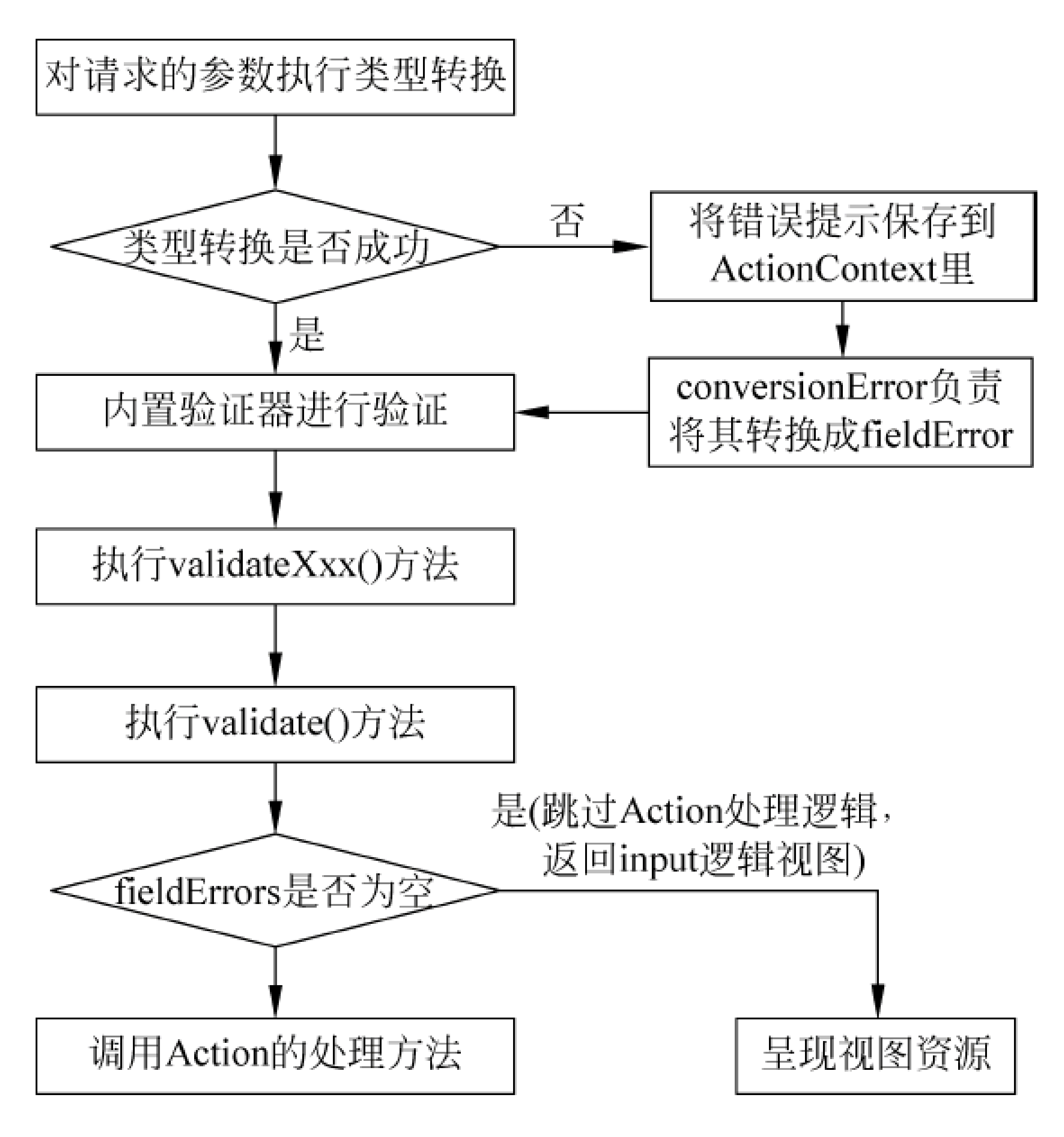


图 13.16　Struts 2 执行输入校验的流程图

13.5　本章小结

本章主要介绍了 Struts 2 的输入校验体系。本章从输入校验的必要性讲起,大致介绍了客户端校验和服务器校验及两种校验的作用。本章重点讲解了手动校验和使用内置校验器实现校验的方法,介绍了常用内置校验器的说明、配置参数,详细讲解了 Visitor 校验器的使用。本章还讲解了校验的流程和顺序,并介绍了 Struts 2 核心组件——值栈。

第14章

在 Struts 2 框架中访问 Web 元素

Struts 2 框架的 Action 未与任何 Servlet API 耦合，这样做的好处是，可以完全脱离 Web 容器测试 Action。当 Action 需要访问 Web 元素时，可以采用两种方法：一种是通过 ActionContext 访问 Servlet API；另一种是通过实现 * Aware()接口来获得 Servlet API。本章我们来学习如何使用这两种方法访问 Servlet API。

本章要点：

- 访问 Web 元素的两种方法
- ActionContext 的概念
- ActionContext 的解读方式

14.1 访问 Web 元素概述

在 Web 应用中，我们一定需要访问 Web 元素(包括 JSP 的内置对象中的 request 对象、session 对象和 application 对象)。例如，当用户登录系统后，为设置用户处于登录状态，通常的作法是向 session 里放一个值，这个值可能是用户的 ID 或用户名、用户权限等信息。当用户进行某一操作时，需要读取 session 中的这个值，来判断用户是否有进行此操作的权限。

Servlet API 中的 HttpServletRequest、HttpServletSession 和 ServletContext，分别代表 JSP 内置对象中的 request 对象、session 对象和 application 对象。

而 Struts 2 框架对 Struts 1 框架进行改良，使 Action 并未直接与任何 Servlet API 耦合，从而方便了 Action 类做单元测试。当 Action 需要访问 Servlet API 时，可以采用两种方法：一种是通过 ActionContext 访问 Servlet API；另一种是通过实现 * Aware()接口来获得 Servlet API。

14.2 通过 ActionContext 访问 Web 元素

14.2.1 ActionContext 简介

Struts 2 框架提供了一个名为 ActionContext 的类，在 Action 中可以通过调用该类的方法来获取 Servlet API。ActionContext 是一个 Action 的上下文

对象，一个处理请求的 Action 是一个线程，这个线程运行期间所涉及的所有数据都保存在 ActionContext 中(如 Session，客户端提交的参数等信息)。

如果把 Action 比喻为一个走在路上的人，那么 ActionContext 就像是记录路况信息的对象，路上的路灯是否亮着，路是柏油马路还是水泥路面，Action 这个“行人”的前后左右是否有车辆行驶，如果有车辆行驶，车辆行驶的方向是什么、速度是多少，等等。

Struts 2 的 Action 可以通过 ActionContext 类的方法来访问 Servlet API。ActionContext 类的常用方法说明如下。

(1) Static ActionContext getContext()：静态方法，该方法返回当前线程的 ActionContext 的对象。

(2) Map getApplication()：返回一个 Map 对象，该对象模拟了该应用的 ServletContext 的实例。

(3) Object get(Object key)：通过参数 key 来查找当前 ActionContext 中的值。该方法可获取 HttpSevletRequest 的属性。

(4) Map getParameters()：获取所有的请求参数，类似于调用 HttpSevletRequest 对象的 get ParametersMap 方法。

(5) Map getSession()：返回一个 Map 对象，该对象模拟了该应用的 HttpServletSession 的实例。

(6) Void put(Object key，Object value)：向当前 ActionContext 对象中存入属性，key 和 value 对应 Actioncontext 对象中的属性名和属性值。

(7) Void setApplication(Map application)：直接传入一个 Map 实例，将该 Map 实例中的 key 和 value 对转换为 application 的属性名和属性值。

(8) Void setSession(Map session)：直接传入一个 Map 实例，将该 Map 实例中的 key 和 value 对转换为 session 的属性名和属性值。

简单地说，ActionContext 的方法使用 Map 类型的对象，模拟 Servlet API 中的 HttpServletRequest、HttpServletSession 和 ServletContext，使用 set *()方法将 Map 类型的对象转换为 HttpServletRequest、HttpServletSession 和 ServletContext 的属性名和属性值，使用 get *()方法将 HttpServletRequest、HttpServletSession 和 ServletContext 的属性名和属性值转换为 Map 类型的对象。

例如，下面的 Action 代码实现了访问 Servlet API 和向 Servlet API 中的 HttpServletRequest、HttpServletSession 和 ServletContext 设置属性。

```
import java.util.Map;
import com.opensymphony.xwork2.ActionContext;
import com.opensymphony.xwork2.ActionSupport;

public class User_Action1 extends ActionSupport{

    private Map request;                    //定义 Map 类型的对象
    private Map session;                    //定义 Map 类型的对象
    private Map application;                //定义 Map 类型的对象
```

```
    public User_Action1(){
        //获得当前线程的 ActionContext 的对象,赋给 actionContext
        ActionContext   actionContext= ActionContext.getContext();
        //获得 Servlet API 中的 HttpServletRequest 的属性,并将属性存放到对象中
        //将对象强制转换为 Map 类型,并赋给 request
        request = (Map)actionContext.get("request");
        //返回一个 Map 对象,该对象模拟了该应用的 HttpServletSession 的实例,并赋给 session
        session = actionContext.getSession();
        //返回一个 Map 对象,该对象模拟了该应用的 ServletContext 的实例,并赋给 application
        application = actionContext.getApplication();
    }

    public String execute() {
        //设置 Map 类型对象的 key 和 value
        request.put("key_r1","value_r1");
        session.put("key_s1", "value_s1");
        application.put("key_a1","value_a1");
        return "success";
    }
}
```

当页面提交请求,创建 User_Action1 的对象,并调用它的 execute()方法时,首先要执行构造方法 User_Action1(),返回 Map 类型的对象,模拟 Servlet API 中的 HttpServletRequest、HttpServletSession 和 ServletContext 对象,并将这些对象赋给 Action 的属性 request、session、application。

执行"request.put("key_r1","value_r1");"语句,在 request 对象中添加 key-value 对,Servlet API 中的 HttpServletRequest 实例会添加对应的属性名和属性值。

同理,执行"session.put("key_s1", "value_s1");"语句,Servlet API 中的 HttpServletSession 实例会添加对应的属性名和属性值,如图 14.1 所示。

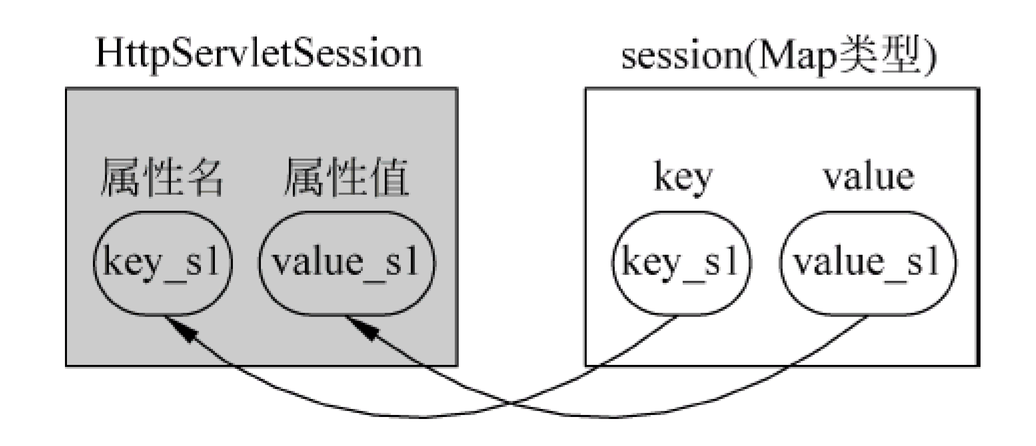


图 14.1　HttpServletSession 与 Map 类型的 session 对象的转换关系

执行"application.put("key_a1","value_a1");"语句,Servlet API 中的 ServletContext 实例会添加对应的属性名和属性值。

在结果视图页可以显示 request 对象、session 对象和 application 对象中的属性值,代码如下:

```
<body>
    显示 Web 元素的值 <br>
    <s:property value="#request.key_r1"/>|
```

```
    <%=request.getAttribute("key_r1") %> <br />
    <s:property value="#session.key_s1"/>|
    <%=session.getAttribute("key_s1") %> <br />
    <s:property value="#application.key_a1"/>
    |<%=application.getAttribute("key_a1") %> <br />
    <s:debug></s:debug>
</body>
```

在浏览器中显示结果如图 14.2 所示。

<s:property value="#session.key_s1"/> 和 <%=session.getAttribute("key_s1") %>效果一样，都可以显示 session 中的属性值。

单击 Debug，显示值栈，可以查看到在 Stack Context 中，session 中有 key_s1=value_s1，如图 14.3 所示。在 Stack Context 中的项目可以使用#Key 的方法获得，#session.key_s1 就获得 session 中 key_s1 的值。



图 14.2 显示访问 Web 元素结果

14.2.2 实践任务 1：通过 ActionContext 访问 Web 元素

1. 任务说明

本任务实践使用 ActionContext 的方法访问 Web 元素。

打开 login.jsp 页面，单击“使用 ActionContext 类访问 Web 元素”超链接，调用 Action 的 execute 方法，访问 Web 元素，向 request 对象中设置属性名为 key_r1、属性值为 value_r1 的属性，向 session 对象中设置属性名为 key_s1、属性值为 value_s1 的属性和向 application 对象中设置属性名为 key_a1、属性值为 value_a1 的属性，打开 index.jsp 页面，显示各属性值，如图 14.2 所示。

2. 任务实施

(1) 创建工程。创建 Struts 2 应用工程 BBS_Struts2_1200_WebElement。

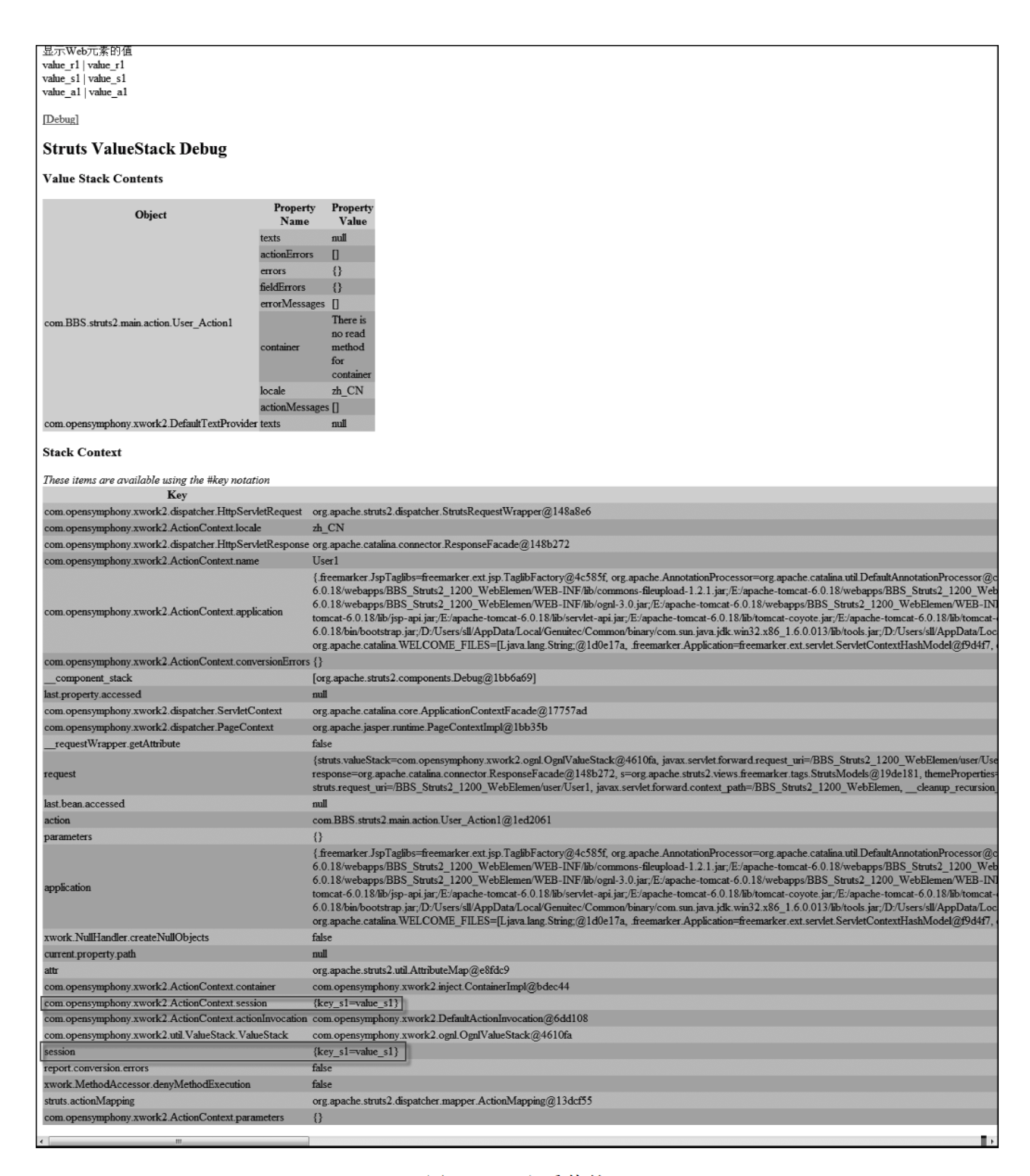
(2) 视图层文件。

① 创建 login.jsp，主要代码如下：

```
<body>
  <a href="">使用 ActionContext 类访问 Web 元素</a>
</body>
```

② 创建 index.jsp，主要代码如下：

```
<body>
    显示 Web 元素的值 <br>
</body>
```



显示Web元素的值
value_r1 | value_r1
value_s1 | value_s1
value_a1 | value_a1

[Debug]

Struts ValueStack Debug

Value Stack Contents

Object	Property Name	Property Value
com.BBS.struts2.main.action.User_Action1	texts	null
	actionErrors	[]
	errors	{}
	fieldErrors	{}
	errorMessages	[]
	container	There is no read method for container
	locale	zh_CN
	actionMessages	[]
com.opensymphony.xwork2.DefaultTextProvider	texts	null

Stack Context

These items are available using the #key notation

Key	
com.opensymphony.xwork2.dispatcher.HttpServletRequest	org.apache.struts2.dispatcher.StrutsRequestWrapper@148a8e6
com.opensymphony.xwork2.ActionContext.locale	zh_CN
com.opensymphony.xwork2.dispatcher.HttpServletResponse	org.apache.catalina.connector.ResponseFacade@148b272
com.opensymphony.xwork2.ActionContext.name	User1
com.opensymphony.xwork2.ActionContext.application	{.freemarker.JspTaglibs=freemarker.ext.jsp.TaglibFactory@4c585f, org.apache.AnnotationProcessor=org.apache.catalina.util.DefaultAnnotationProcessor@c 6.0.18/webapps/BBS_Struts2_1200_WebElemen/WEB-INF/lib/commons-fileupload-1.2.1.jar;/E:/apache-tomcat-6.0.18/webapps/BBS_Struts2_1200_Web 6.0.18/webapps/BBS_Struts2_1200_WebElemen/WEB-INF/lib/ognl-3.0.jar;/E:/apache-tomcat-6.0.18/webapps/BBS_Struts2_1200_WebElemen/WEB-INI tomcat-6.0.18/lib/jsp-api.jar;/E:/apache-tomcat-6.0.18/lib/servlet-api.jar;/E:/apache-tomcat-6.0.18/lib/tomcat-coyote.jar;/E:/apache-tomcat-6.0.18/lib/tomcat- 6.0.18/bin/bootstrap.jar;/D:/Users/sll/AppData/Local/Genuitec/Common/binary/com.sun.java.jdk.win32.x86_1.6.0.013/lib/tools.jar;/D:/Users/sll/AppData/Loc org.apache.catalina.WELCOME_FILES=[Ljava.lang.String;@1d0e17a, .freemarker.Application=freemarker.ext.servlet.ServletContextHashModel@f9d4f7,
com.opensymphony.xwork2.ActionContext.conversionErrors	{}
__component_stack	[org.apache.struts2.components.Debug@1bb6a69]
last.property.accessed	null
com.opensymphony.xwork2.dispatcher.ServletContext	org.apache.catalina.core.ApplicationContextFacade@17757ad
com.opensymphony.xwork2.dispatcher.PageContext	org.apache.jasper.runtime.PageContextImpl@1bb35b
__requestWrapper.getAttribute	false
request	{struts.valueStack=com.opensymphony.xwork2.ognl.OgnlValueStack@4610fa, javax.servlet.forward.request_uri=/BBS_Struts2_1200_WebElemen/user/Use response=org.apache.catalina.connector.ResponseFacade@148b272, s=org.apache.struts2.views.freemarker.tags.StrutsModels@19de181, themeProperties= struts.request_uri=/BBS_Struts2_1200_WebElemen/user/User1, javax.servlet.forward.context_path=/BBS_Struts2_1200_WebElemen, __cleanup_recursion_
last.bean.accessed	null
action	com.BBS.struts2.main.action.User_Action1@1ed2061
parameters	{}
application	{.freemarker.JspTaglibs=freemarker.ext.jsp.TaglibFactory@4c585f, org.apache.AnnotationProcessor=org.apache.catalina.util.DefaultAnnotationProcessor@c 6.0.18/webapps/BBS_Struts2_1200_WebElemen/WEB-INF/lib/commons-fileupload-1.2.1.jar;/E:/apache-tomcat-6.0.18/webapps/BBS_Struts2_1200_Web 6.0.18/webapps/BBS_Struts2_1200_WebElemen/WEB-INF/lib/ognl-3.0.jar;/E:/apache-tomcat-6.0.18/webapps/BBS_Struts2_1200_WebElemen/WEB-INI tomcat-6.0.18/lib/jsp-api.jar;/E:/apache-tomcat-6.0.18/lib/servlet-api.jar;/E:/apache-tomcat-6.0.18/lib/tomcat-coyote.jar;/E:/apache-tomcat-6.0.18/lib/tomcat- 6.0.18/bin/bootstrap.jar;/D:/Users/sll/AppData/Local/Genuitec/Common/binary/com.sun.java.jdk.win32.x86_1.6.0.013/lib/tools.jar;/D:/Users/sll/AppData/Loc org.apache.catalina.WELCOME_FILES=[Ljava.lang.String;@1d0e17a, .freemarker.Application=freemarker.ext.servlet.ServletContextHashModel@f9d4f7,
xwork.NullHandler.createNullObjects	false
current.property.path	null
attr	org.apache.struts2.util.AttributeMap@e8fdc9
com.opensymphony.xwork2.ActionContext.container	com.opensymphony.xwork2.inject.ContainerImpl@bdec44
com.opensymphony.xwork2.ActionContext.session	{key_s1=value_s1}
com.opensymphony.xwork2.ActionContext.actionInvocation	com.opensymphony.xwork2.DefaultActionInvocation@6dd108
com.opensymphony.xwork2.util.ValueStack.ValueStack	com.opensymphony.xwork2.ognl.OgnlValueStack@4610fa
session	{key_s1=value_s1}
report.conversion.errors	false
xwork.MethodAccessor.denyMethodExecution	false
struts.actionMapping	org.apache.struts2.dispatcher.mapper.ActionMapping@13dcf55
com.opensymphony.xwork2.ActionContext.parameters	{}

图 14.3　查看值栈

（3）创建 Action 类。创建 User_Action1，代码如下：

```
package com.BBS.struts2.main.action;

import java.util.Map;
import com.opensymphony.xwork2.ActionContext;
import com.opensymphony.xwork2.ActionSupport;

public class User_Action1 extends ActionSupport{
    private Map request;
```

```
    private Map session;
    private Map application;

    public User_Action1(){
        //获得当前线程的 ActionContext 的对象,赋给 actionContext
        ActionContext  actionContext= ActionContext.getContext();
        //获得 Map 类型的 request 对象
        request = (Map)actionContext.get("request");
        //获得 Map 类型的 session 对象
        session = actionContext.getSession();
        //获得 Map 类型的 application 对象
        application = actionContext.getApplication();
    }

    public String execute() {
        //向 Map 对象中设置值
        request.put("key_r1","value_r1");
        session.put("key_s1", "value_s1");
        application.put("key_a1","value_a1");
        return "success";
    }
}
```

(4) 配置 Action。打开配置文件 struts.xml,输入以下代码。

```
<package name="user" namespace="/user" extends="struts-default">
<action name="User*" class="com.BBS.struts2.main.action.User_Action{1}">
    <result>/main/index.jsp </result>
</action>
</package>
```

(5) 编写视图层,提交请求,并设置 Web 元素信息。

① 打开 login.jsp,代码如下:

```
<body>
  <a href="user/User1">使用 ActionContext 类访问 Web 元素</a>
</body>
```

② 打开 index.jsp,代码如下:

```
<body>
    显示 Web 元素的值 <br>
    <s:property value="#request.key_r1"/> |
    <%=request.getAttribute("key_r1") %> <br />
    <s:property value="#session.key_s1"/> |
    <%=session.getAttribute("key_s1") %> <br />
    <s:property value="#application.key_a1"/> |
    <%=application.getAttribute("key_a1") %> <br />
    <s:debug></s:debug>
</body>
```

（6）测试。部署工程，在浏览器的地址栏中输入 http://localhost:8080/BBS_Struts2_1200_WebElemen/main/login.jsp，打开 login.jsp 页面，单击“使用 ActionContext 类访问 Web 元素”超链接，打开 index.jsp 页面，显示 Web 元素属性值，如图 14.2 所示。

单击<s:debug>标签，打开 ValueStack，查看 request、session 和 application 中的值，如图 14.3 所示。

3. 任务总结

本任务通过 ActionContext 访问 Web 元素，获得 request、session 和 application 对象，并设置其属性。

14.2.3　实践任务 2：通过 ActionContext 访问 Web 元素设置登录状态

1. 任务说明

本任务实践使用 ActionContext 的方法访问 Web 元素，设置用户登录状态。

打开 login.jsp 页面（见图 14.4），输入用户名和密码，单击“登录”按钮，调用 Action 的 execute 方法，访问 Web 元素，将用户名放置到 session 对象中（表示该用户处于登录状态），打开 index.jsp 页面，显示登录用户名，如图 14.5 所示。

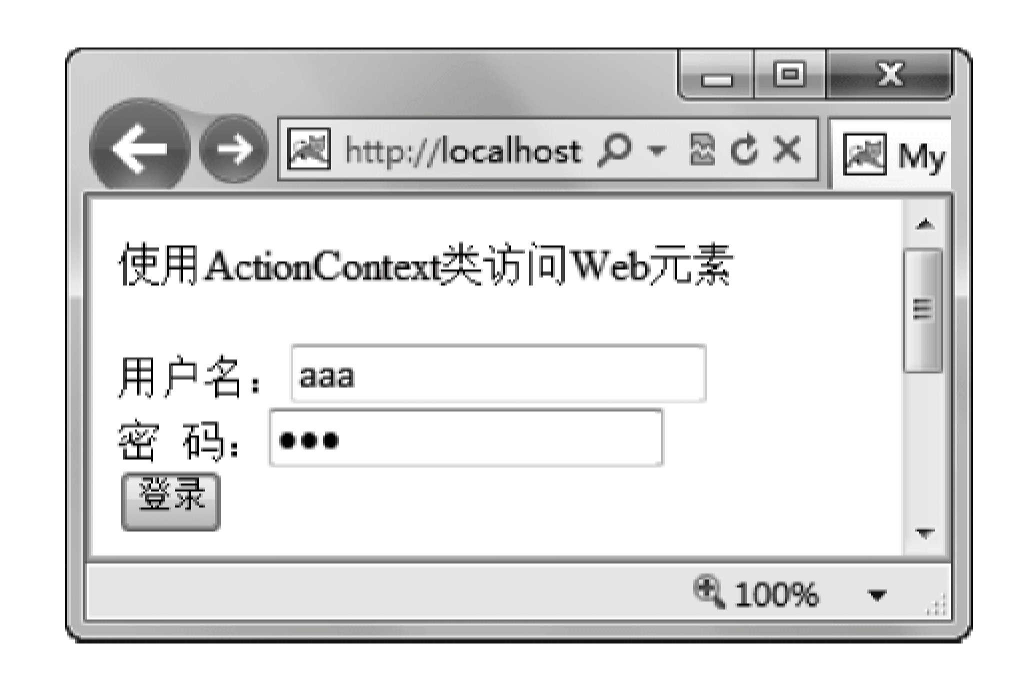


图 14.4　login.jsp 页面



图 14.5　显示登录用户名

2. 任务实施

（1）创建工程。复制工程 BBS_Struts2_1200_WebElement，并粘贴，创建 Struts 2 应用工程 BBS_Struts2_1250_WebElement。

（2）视图层文件。

① 修改 login.jsp，主要代码如下：

```
<body>
  使用 ActionContext 类访问 Web 元素
   <form action="user/User1" method="get">
       用户名：<input type="text" /><br>
       密　码：<input type="password" /><br>
       <input type="submit" value="登录">
   </form>
```

```
</body>
```

② 修改 index.jsp，主要代码如下：

```
<body>
    登录成功!登录用户的用户名为: <br>
</body>
```

(3) 修改 Action 类。创建 User_Action1，代码如下：

```
package com.BBS.struts2.main.action;

import java.util.Map;
import com.opensymphony.xwork2.ActionContext;
import com.opensymphony.xwork2.ActionSupport;

public class User_Action1 extends ActionSupport{
    private String name;                    //用于接收请求参数
    private Map session;                    //用于访问 Web 元素

    public User_Action1(){
        //获得当前线程的 ActionContext 的对象,赋给 actionContext
        ActionContext  actionContext= ActionContext.getContext();
        //获得 Map 类型的 session 对象
        session = actionContext.getSession();
    }

    public String execute() {
        //向 Map 对象中设置值
        session.put("username", name);
        return "success";
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

(4) 配置 Action。Action 的配置信息不必修改。

(5) 编写视图层，提交请求，并设置 Web 元素信息。

① 打开 login.jsp，代码如下：

```
<body>
   使用 ActionContext 类访问 Web 元素
   <form action="user/User1" method="get">
       用户名: <input type="text"   name="name"/><br>
```

```
        密　码：<input type="password" /><br>
        <input type="submit" value="登录">
    </form>
</body>
```

② 打开 index.jsp，代码如下：

```
<body>
    登录成功！登录用户的用户名为：<s:property value="#session.username"/><br />
    <s:debug></s:debug>
</body>
```

（6）测试。部署工程，在浏览器的地址栏中输入 http://localhost:8080/BBS_Struts2_1250_WebElemen/main/login.jsp，打开 login.jsp 页面，输入用户名和密码，单击“登录”按钮，打开 index.jsp 页面，显示登录用户名，如图 14.5 所示。

单击<s:debug>标签，打开 ValueStack，查看 session 的属性 username=aaa，如图 14.6 所示。

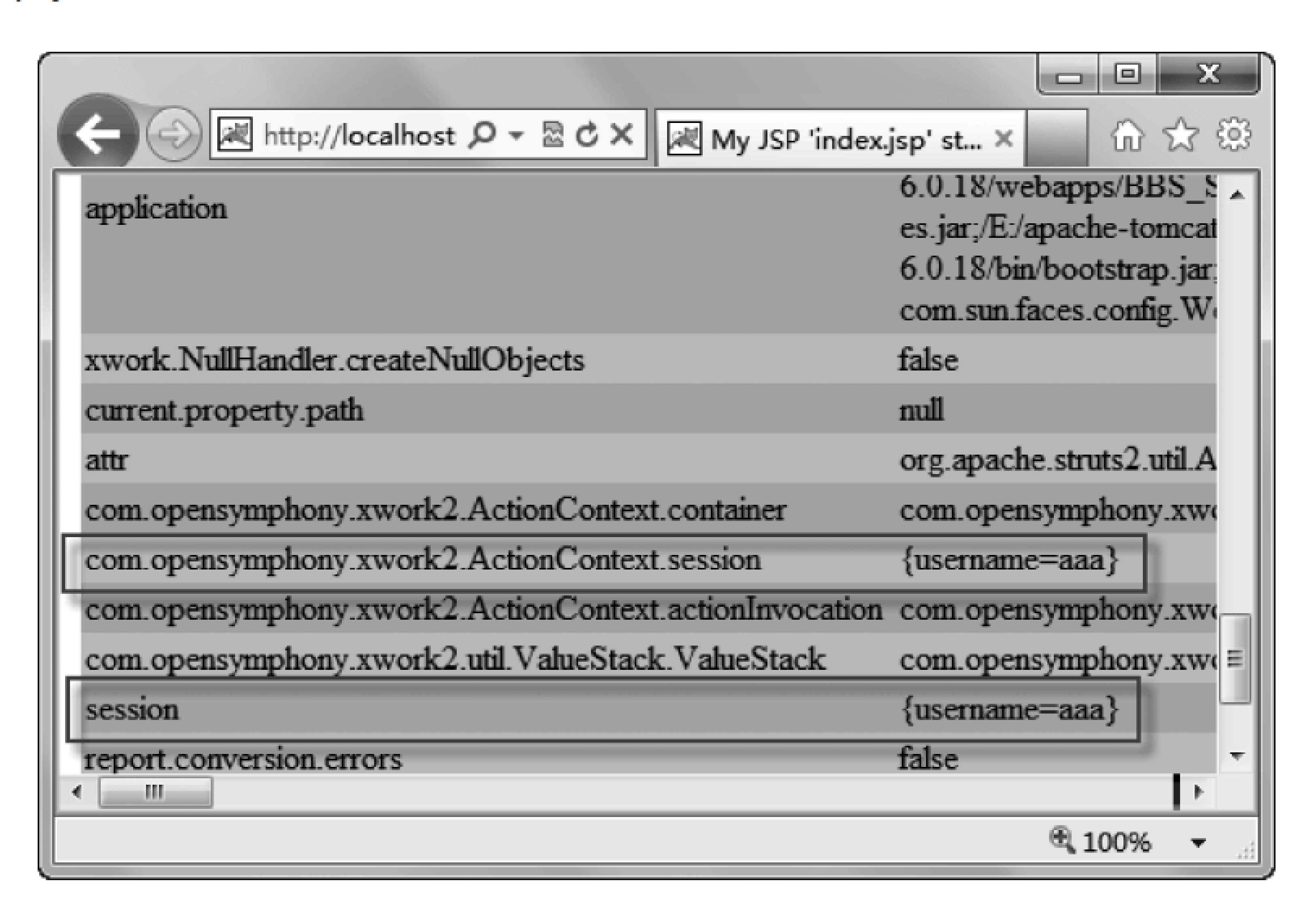


图 14.6　查看 session 中的属性

3. 任务总结

本任务通过 ActionContext 访问 Web 元素，获得 session 对象，将登录用户的用户名设置到 session 中。

14.3　通过实现 * Aware 接口访问 Web 元素

14.3.1　* Aware 简介

Action 还可以通过使用 * Aware 接口的方式直接访问 Servlet API。比如，要获得

session 对象，就可以让 Action 继承 SessionAware 接口，并实现其抽象方法 setSession()。除了 session 对象，Action 当然还可以获得其他对象，如表 14.1 所示。

表 14.1　可获得的常用对象

接口名称	获得 Servlet 对象的方法
RequestAware	void setRequest(Map<String, Object> request)
SessionAware	void setSession(Map<String, Object> session)
ApplicationAware	void setApplication(Map<String, Object> application)

Struts 在实例化一个 Action 中，如果发现它实现了相应的 Aware 接口，会把相应的资源通过 Aware 接口方法注入进去，这种方法可以叫做注入方式(IoC 方式)。

例如，下面的 Action 代码实现了访问 Servlet API 和向 Servlet API 中的 HttpServletRequest、HttpServletSession 和 ServletContext 设置属性。

```
import java.util.Map;
import com.opensymphony.xwork2.ActionSupport;
import org.apache.struts2.interceptor.ApplicationAware;
import org.apache.struts2.interceptor.RequestAware;
import org.apache.struts2.interceptor.SessionAware;

public class User_Action2 extends ActionSupport implements RequestAware, SessionAware,
ApplicationAware {

    private Map<String, Object> request;
    private Map<String, Object> session;
    private Map<String, Object> application;

    public String execute() {
        request.put("key_r1","value_r1");
        session.put("key_s1", "value_s1");
        application.put("key_a1","value_a1");
        return "success";
    }

    public void setSession(Map<String, Object> session) {
        this.session=session;
    }
    public void setApplication(Map<String, Object> application) {
        this.application=application;
    }
    public void setRequest(Map<String, Object> request) {
        this.request=request;
    }
}
```

当页面提交请求时，创建 User_Action2 的对象并调用它的 execute()方法时，Struts 2 框架发现 User_Action2 实现了 RequestAware、SessionAware 和 ApplicationAwarerequest 接口，

则将相应资源注入 Action 对象中，Action 对象就可以直接访问 Servlet API。

在结果视图页可以显示 request 对象、session 对象和 application 对象中的属性值，代码同前。在浏览器中的显示结果如图 14.2 所示。

单击 Debug，显示值栈，可以查看到在 Stack Context 中，session 中有 key_s1=value_s1，如图 14.3 所示。

14.3.2　实践任务 3：通过实现 * Aware 接口访问 Web 元素

1. 任务说明

本任务实践通过实现 * Aware 接口访问 Web 元素。

打开 login.jsp 页面，单击“使用接口访问 Web 元素”超链接，调用 Action 的 execute 方法，访问 Web 元素，向 request 对象中设置属性名为 key_r1、属性值为 value_r1 的属性，向 session 对象中设置属性名为 key_s1、属性值为 value_s1 的属性和向 application 对象中设置属性名为 key_a1、属性值为 value_a1 的属性，打开 index.jsp 页面，显示各属性值，如图 14.2 所示。

2. 任务实施

(1) 打开工程。打开 Struts 2 应用工程 BBS_Struts2_1200_WebElement。

(2) 视图层文件。修改 login.jsp，主要代码如下：

```
<body>
  <a href="">使用 ActionContext 类访问 Web 元素</a>
  <a href="">使用接口访问 Web 元素</a>
</body>
```

(3) 创建 Action 类。创建 User_Action2，代码如下：

```
import java.util.Map;
import com.opensymphony.xwork2.ActionSupport;
import org.apache.struts2.interceptor.ApplicationAware;
import org.apache.struts2.interceptor.RequestAware;
import org.apache.struts2.interceptor.SessionAware;

public class User_Action2 extends ActionSupport implements RequestAware, SessionAware,
ApplicationAware {

    private Map<String, Object> request;
    private Map<String, Object> session;
    private Map<String, Object> application;

    public String execute() {
        request.put("key_r1","value_r1");
        session.put("key_s1", "value_s1");
        application.put("key_a1","value_a1");
        return "success";
```

```
    }

    public void setSession(Map<String, Object> session) {
        this.session=session;
    }
    public void setApplication(Map<String, Object> application) {
        this.application=application;
    }
    public void setRequest(Map<String, Object> request) {
        this.request=request;
    }
}
```

(4) 配置 Action。配置 Action 的代码不需要修改。

(5) 编写视图层，提交请求，并设置 Web 元素信息。

① 打开 login.jsp，代码如下：

```
<body>
  <a href="user/User1">使用 ActionContext 类访问 Web 元素</a>
  <a href="user/User2">使用接口访问 Web 元素</a>
</body>
```

② index.jsp 代码不必修改。

(6) 测试。部署工程，在浏览器的地址栏中输入 http://localhost:8080/BBS_Struts2_1200_WebElement/main/login.jsp，打开 login.jsp 页面，单击"使用接口访问 Web 元素"超链接，打开 index.jsp 页面，显示 Web 元素属性值，如图 14.2 所示。

单击<s:debug>标签，打开 ValueStack，查看 request、session 和 application 中的值，如图 14.3 所示。

3. 任务总结

本任务通过实现接口访问 Web 元素，获得 request、session 和 application 对象，并设置其属性。

14.3.3 实践任务 4：使用接口访问 Web 元素，实现退出系统

1. 任务说明

本任务实践使用接口访问 Web 元素，使用户退出系统。

当用户登录系统后，打开 index.jsp 页面(见图 14.7)，单击"退出"超链接，调用 Action 的 excute 方法，访问 Web 元素，将 session 对象中的用户名删除，返回到 login.jsp 页面。



图 14.7 登录成功显示的信息

2. 任务实施

(1) 打开工程。打开工程 BBS_Struts2_1250_WebElement。

(2) 视图层文件。修改 index.jsp，主要代码如下：

```
<body>
    显示 Web 元素的值 <br>
    登录成功!登录用户的用户名为: <s:property value="#session.username"/><br />
    <a href="">退出</a>
    <s:debug></s:debug>
</body>
```

(3) 创建 Action 类。创建 Quit_Action，代码如下：

```
package com.BBS.struts2.main.action;
import java.util.Map;
import org.apache.struts2.interceptor.SessionAware;
import com.opensymphony.xwork2.ActionSupport;

public class Quit_Action extends ActionSupport implements SessionAware{
    private Map<String, Object> session;
    public String execute() {
        session.remove("username");
        return "success";
    }

    public void setSession(Map<String, Object> session) {
        this.session=session;
    }
}
```

(4) 配置 Action。打开配置文件 struts.xml，输入以下代码：

```
<package name="user" namespace="/user" extends="struts-default">
    <action name="User*" class="com.BBS.struts2.main.action.User_Action{1}">
        <result> /main/index.jsp </result>
    </action>
    <action name="Quit" class="com.BBS.struts2.main.action.Quit_Action">
        <result>/main/login.jsp</result>
    </action>
</package>
```

(5) 编写视图层，提交请求，插入 debug 标签。

① 打开 login.jsp，插入 debug 标签，代码如下：

```
<body>
   使用 ActionContext 类访问 Web 元素
   <form action="user/User1" method="get">
       用户名: <input type="text"  name="name"/><br>
```

```
        密　码：<input type="password" /><br>
        <input type="submit" value="登录">
    </form>
<s:debug></s:debug>
</body>
```

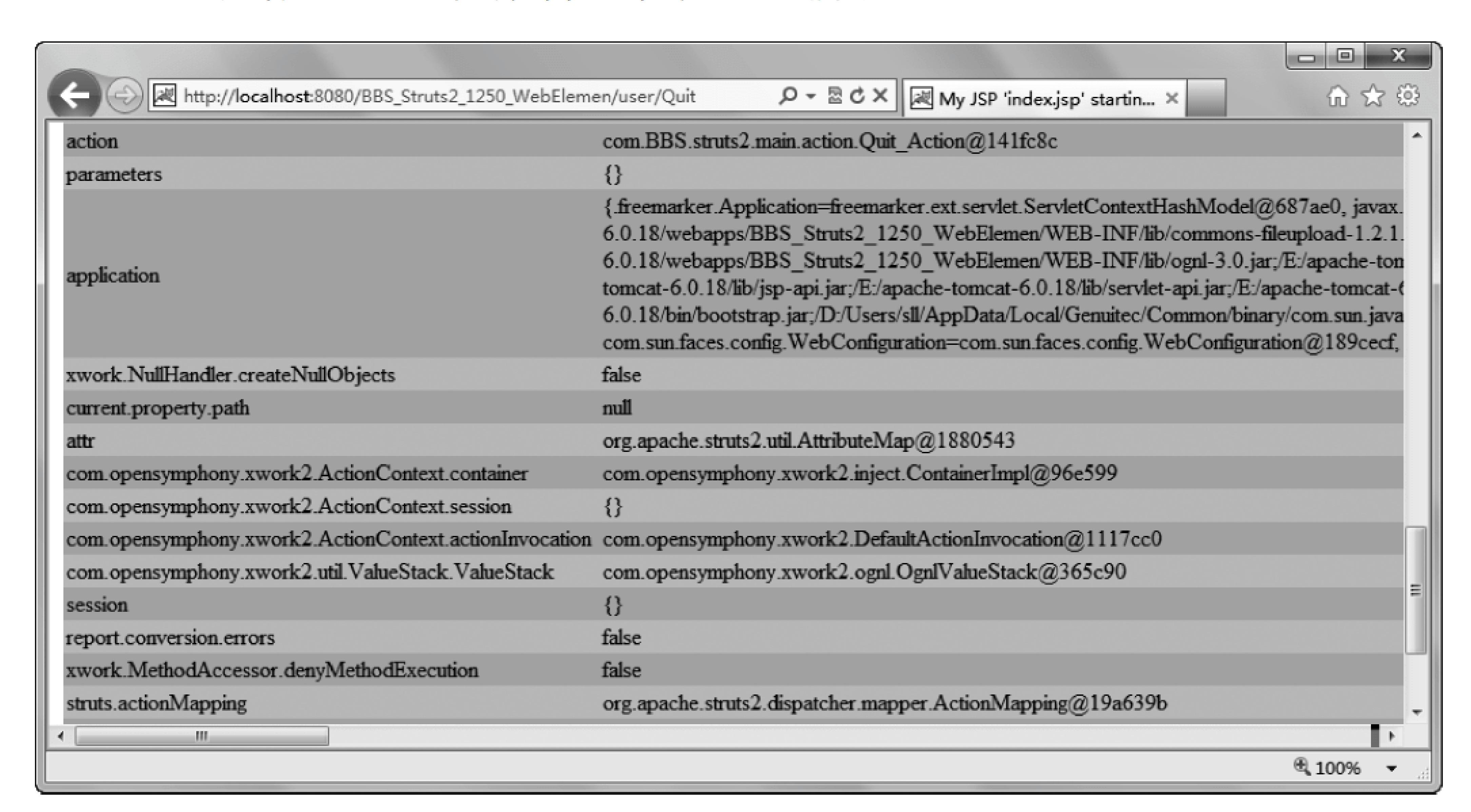
② 打开 index.jsp，代码如下：

```
<body>
    显示 Web 元素的值 <br>
    登录成功!登录用户的用户名为：<s:property value="#session.username"/><br/>
    <a href="user/Quit">退出</a>
    <s:debug></s:debug>
</body>
```

(6) 测试。部署工程，在浏览器的地址栏中输入 http://localhost:8080/BBS_Struts2_1250_WebElemen/main/login.jsp，打开 login.jsp 页面，输入用户名和密码，单击“登录”按钮，打开 index.jsp 页面，显示登录用户名，如图 14.4 所示。

单击<s:debug>标签，打开 ValueStack，查看 session 的属性 username=aaa，如图 14.7 所示。

单击“退出”超链接，返回 login.jsp 页面。单击<s:debug></s:debug>标签，打开 ValueStack，查看 session 的属性为空，如图 14.8 所示。



action	com.BBS.struts2.main.action.Quit_Action@141fc8c
parameters	{}
application	{freemarker.Application=freemarker.ext.servlet.ServletContextHashModel@687ae0, javax. 6.0.18/webapps/BBS_Struts2_1250_WebElemen/WEB-INF/lib/commons-fileupload-1.2.1. 6.0.18/webapps/BBS_Struts2_1250_WebElemen/WEB-INF/lib/ognl-3.0.jar;/E:/apache-ton tomcat-6.0.18/lib/jsp-api.jar;/E:/apache-tomcat-6.0.18/lib/servlet-api.jar;/E:/apache-tomcat-6 6.0.18/bin/bootstrap.jar;/D:/Users/sll/AppData/Local/Genuitec/Common/binary/com.sun.java com.sun.faces.config.WebConfiguration=com.sun.faces.config.WebConfiguration@189cecf,
xwork.NullHandler.createNullObjects	false
current.property.path	null
attr	org.apache.struts2.util.AttributeMap@1880543
com.opensymphony.xwork2.ActionContext.container	com.opensymphony.xwork2.inject.ContainerImpl@96e599
com.opensymphony.xwork2.ActionContext.session	{}
com.opensymphony.xwork2.ActionContext.actionInvocation	com.opensymphony.xwork2.DefaultActionInvocation@1117cc0
com.opensymphony.xwork2.util.ValueStack.ValueStack	com.opensymphony.xwork2.ognl.OgnlValueStack@365c90
session	{}
report.conversion.errors	false
xwork.MethodAccessor.denyMethodExecution	false
struts.actionMapping	org.apache.struts2.dispatcher.mapper.ActionMapping@19a639b

图 14.8　查看退出系统后 session 中的属性

3. 任务总结

本任务使用接口 Web 元素，获得 session 对象，将 session 中的用户名移除，退出系统。

14.4　实训 1：实现论坛管理系统中设置用户登录状态

1. 要求

使用实现接口方式访问 Web 元素，实现设置用户登录状态，当用户登录系统后，将用户名放在 session 中。

2. 建议步骤

（1）使用工程 BBS_Struts2_1250_WebElement。

（2）编辑视图层文件 login.jsp，创建登录表单。

（3）创建 Action，使用接口实现访问 Web 元素，当用户登录后将用户名放在 session 中。

（4）编写 Action 配置。

（5）编辑视图层文件 login.jsp，提交登录请求。index.jsp 页面不必修改。

（6）测试。

14.5　实训 2：实现论坛管理系统中退出系统功能

1. 要求

使用通过 ActionContext 访问 Web 元素，将 session 中的用户名移除，返回登录页面，实现退出系统的功能。

2. 建议步骤

（1）使用工程 BBS_Struts2_1250_WebElement。

（2）创建 Action，通过 ActionContext 访问 Web 元素，将 session 中的用户名移除。

（3）编写 Action 配置。

（4）编辑视图层文件 index.jsp，提交退出系统的请求。login.jsp 页面不必修改。

（5）测试。

14.6　本章小结

本章主要介绍了 Struts 2 访问 Web 元素的方法，包括通过 ActionContext 访问和通过实现 * Aware 接口的方式访问。

第15章

Struts 2 的结果类型

Action 处理完用户请求后，返回逻辑视图，这个逻辑视图就是一个普通的字符串。然后 Struts 2 框架根据配置文件 struts.xml 中逻辑视图名和物理视图之间的映射关系，将对应的物理视图呈现给用户。

除此之外，Struts 2 框架还支持多种结果映射：Struts 2 框架将处理结果转向实际资源时，实际资源不仅可以是 JSP 视图资源，也可以是 FreeMarker 视图资源，还可以将请求转给另一个 Action 处理，形成 Action 的链式处理。

Struts 2 框架还可以将共用的结果设置为全局结果，全局结果的作用范围是对所有的 Action 都有效。

本章要点：

- 常用结果类型
- 全局结果

15.1 常用结果类型

Struts 2 支持多种结果类型，在 struts-default.xml 文件中，我们可以找到 Struts 2 所支持的结果类型。struts-default.xml 文件在 struts2-core-2.2.1.jar 包中，双击可以看到其中的代码，如下所示。

```
<result-types>
  <result-type name="chain"
               class="com.opensymphony.xwork2.ActionChainResult"/>
  <result-type name="dispatcher"
               class = " org. apache. struts2. dispatcher. ServletDispatcherResult "
               default="true"/>
  <result-type name="freemarker"
               class="org.apache.struts2.views.freemarker.FreemarkerResult"/>
  <result-type name="httpheader"
               class="org.apache.struts2.dispatcher.HttpHeaderResult"/>
  <result-type name="redirect"
               class="org.apache.struts2.dispatcher.ServletRedirectResult"/>
  <result-type name="redirectAction"
               class="org.apache.struts2.dispatcher.ServletActionRedirectResult"/>
  <result-type name="stream"
               class="org.apache.struts2.dispatcher.StreamResult"/>
  <result-type name="velocity"
               class="org.apache.struts2.dispatcher.VelocityResult"/>
```

```
    <result-type name="xslt" class="org.apache.struts2.views.xslt.XSLTResult"/>
    <result-type name="plainText" class="org.apache.struts2.dispatcher.PlainTextResult" />
</result-types>
```

struts-default.xml 是 Struts2 框架的默认配置文件，struts.xml 文件中<package>元素的属性 extends="struts-default"表示继承该配置文件。

result 标签可以通过 type 属性，指定结果类型。其结果类型的作用如表 15.1 所示。

表 15.1　结果类型

结果类型名称	设 置 格 式	结果类型功能描述
Chain Result	type="chain"	用来处理 Action 链
Dispatcher Result	type="dispatcher"	用来转向页面，通常处理 JSP
Redirect Result	type="redirect"	重定向到一个 URL
Redirect Action Result	type="redirectAction"	重定向到一个 Action
FreeMarker Result	type="freemarker"	处理 FreeMarker 模板
HttpHeader Result	type="httpheader"	用来控制特殊的 Http 行为
Stream Result	type="stream"	向发送 InputStream 对象，通常用来处理文件下载
Velocity Result	type="velocity"	处理 Velocity 模板
XLST Result	type="xslt"	处理 XML/XLST 模板
PlainText Result	type="plainText"	显示原始文件内容，例如文件源代码

虽然 Struts 2 框架支持的结果类型很多，但在实际应用中最常用的是 dispatcher，也经常会用到 redirect，有时会用到 chain 和 redirectAction，其他结果类型较少使用。下面我们详细介绍这四个常用的结果类型。

15.1.1　dispatcher 结果类型

dispatcher 结果类型是将请求转发到指定的视图资源(通常为 JSP 资源)。它是默认的结果类型，像我们之前配置<action>元素时，省略了 type 属性的设置，其结果类型均为 dispatcher。

dispatcher 结果类型有以下特点。

(1) 转发的结果是一个视图资源(通常为 JSP 资源)。

(2) 只能将请求转发至同一个 Web 应用中。

(3) 利用请求转发浏览器地址栏不会发生变化。

(4) 利用请求转发调用者和被调用者属于同一个访问请求和响应，两者之间共享相同的 request 对象和 response 对象，且请求参数、请求属性、Action 实例和 Action 中封装的属性都保留。

15.1.2　redirect 结果类型

redirect 结果类型与 dispatcher 结果类型相似，两者都常用于处理 JSP 视图资源，区

别在于 dispatcher 结果类型是请求转发，而 redirect 结果类型是将请求重定向到指定的视图资源。

redirect 结果类型有以下特点。

（1）重定向的结果是一个视图资源（通常为 JSP 资源）。

（2）重定向不仅可以指定到同一个 Web 应用中，还能够指定到任何 JSP 资源。

（3）重定向的访问结束后，浏览器地址栏会显示 URL 的变化。

（4）利用请求转发调用者和被调用者属于两个独立的访问请求和响应过程，两者使用各自的 request 对象和 response 对象。

（5）重定向是产生一个新的请求，因此所有的请求参数、请求属性、Action 实例和 Action 中封装的属性全部丢失。

15.1.3　实践任务 1：比较 dispatcher 结果类型和 redirect 结果类型

1. 任务说明

本任务实践 dispatcher 结果类型和 redirect 结果类型的使用。

分别使用 dispatcher 结果类型和 redirect 结果类型实现系统登录，观察两者的异同。

2. 任务实施

（1）创建工程。创建工程 BBS_Struts2_1300_resultType。

（2）视图层文件。

① 创建 login.jsp，主要代码如下：

```
<body>
   <form action=" " method="get">
       用户名：<input type="text"   name="name"/><br>
       密　码：<input type="password" name="password"/><br>
       <input type="submit" value="登录">
   </form>
</body>
```

② 创建 index.jsp，主要代码如下：

```
<body>
    登录成功!欢迎×××登录本系统!<br/>
</body>
```

（3）创建 Action 类。创建 User_Action，代码如下：

```
package com.BBS.struts2.main.action;
import com.opensymphony.xwork2.ActionSupport;

public class User_Action extends ActionSupport{
    private String name;
    private String password;
```

```
    public String login() {
        return SUCCESS;
    }
    //属性的 getter 和 setter 方法略
    ...
}
```

(4) 配置 Action。打开配置文件 struts.xml,输入以下代码。

```
<package name="user" namespace="/user" extends="struts-default">
  <action name="User_login" class="com.BBS.struts2.main.action.User_Action"
        method="login">
      <result>/index.jsp</result>
  </action>
</package>
```

未设置结果类型,则结果类型默认为 dispatcher。

(5) 编写视图层,提交请求,插入 debug 标签。

① 打开 login.jsp,提交请求,代码如下:

```
<body>
   <form action="user/User_login" method="get">
       用户名: <input type="text"  name="name"/><br>
       密  码: <input type="password" name="password"/><br>
       <input type="submit" value="登录">
   </form>
</body>
```

② 打开 index.jsp,代码如下:

```
<body>
    登录成功!欢迎<s:property value="name"/>登录本系统!<br />
    <s:debug></s:debug>
</body>
```

(6) 测试。部署工程,在浏览器的地址栏中输入 http://localhost:8080/BBS_Struts2_1300_resultType/login.jsp,打开 login.jsp 页面,输入用户名和密码(见图 15.1),单击“登录”按钮,打开 index.jsp 页面,显示登录用户名,如图 15.2 所示。可以看到虽然页面已经跳转到 index.jsp,而地址栏还显示请求的 URL: http://localhost:8080/BBS_Struts2_1300_resultType/user/User_login?name=aaa&password=123。

单击<s:debug>标签,打开 ValueStack,查看 Action 的实例和 Action 中封装的属性 name、password 等都保留,因此用户名“aaa”显示在页面中,如图 15.3 所示。

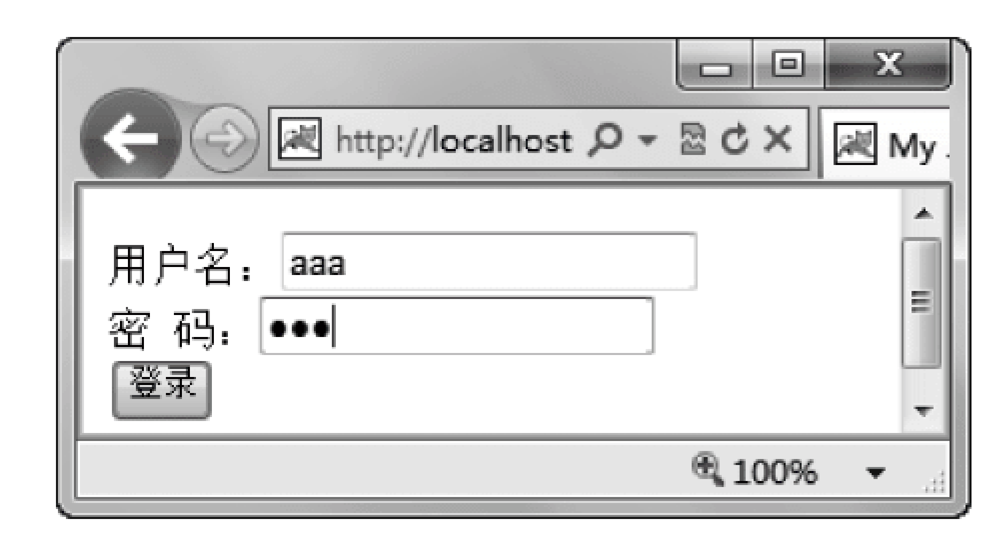


图 15.1　login.jsp 页面

(7) 修改 Action 配置。打开配置文件 struts.xml,修改结果类型为 redirect,代码

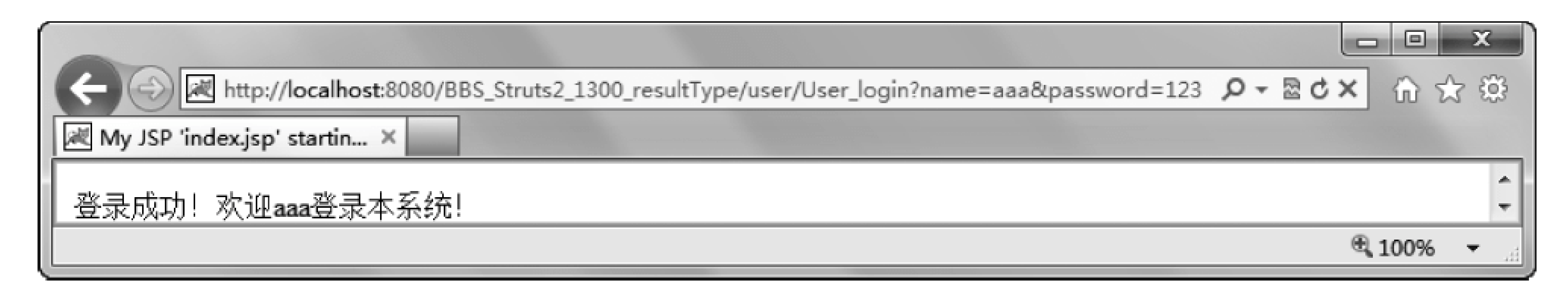


图 15.2　请求转发到 index.jsp 页面

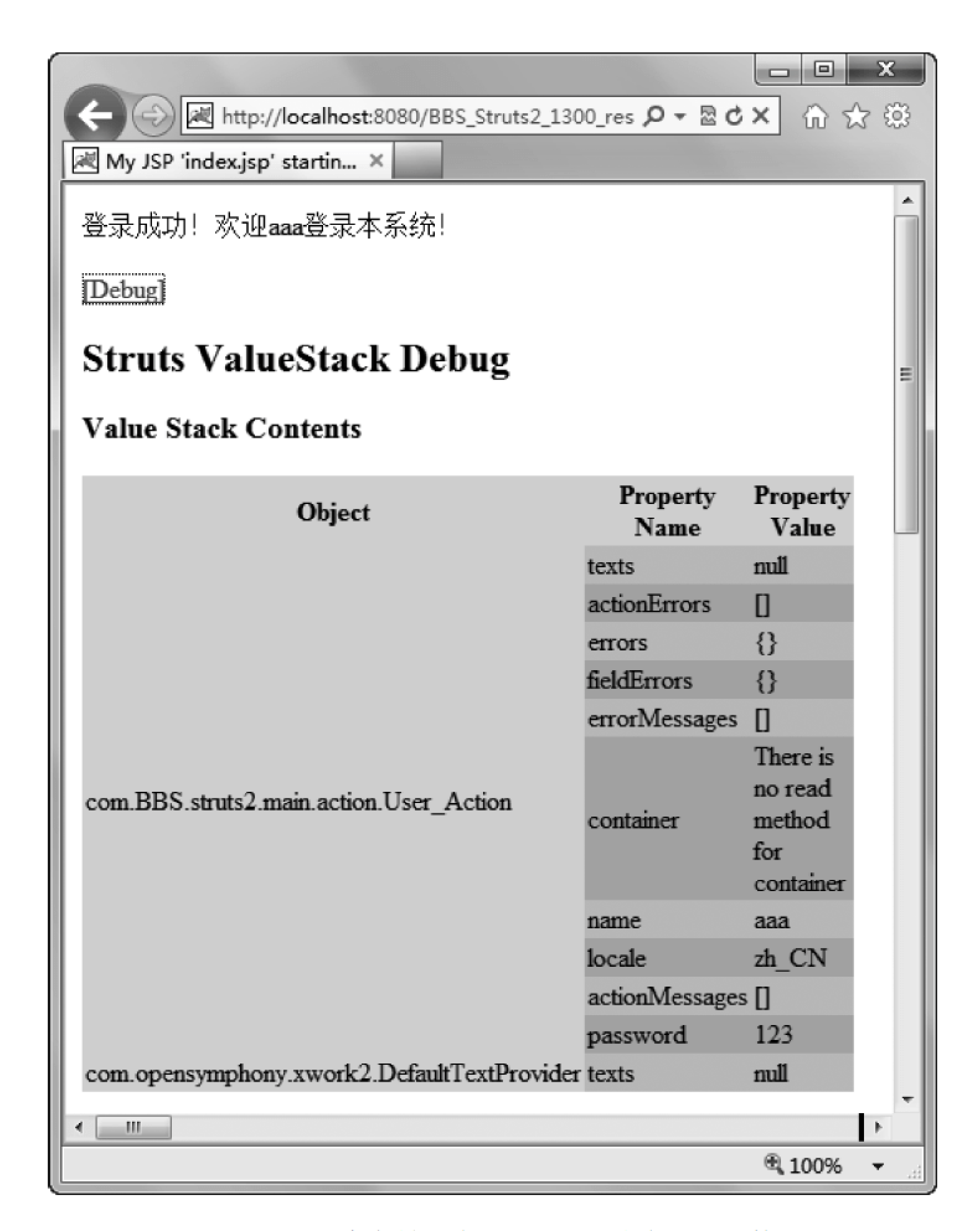


图 15.3　请求转发保留 Action 实例、属性等

如下：

```
<package name="user" namespace="/user" extends="struts-default">
    <action name="User_login" class="com.BBS.struts2.main.action.User_Action"
            method="login">
        <result type="redirect">/index.jsp</result>
    </action>
</package>
```

(8) 测试。部署工程，在浏览器的地址栏中输入 http://localhost:8080/BBS_

Struts2_1300_resultType/login.jsp，打开 login.jsp 页面，输入用户名和密码（见图 15.1），单击“登录”按钮，打开 index.jsp 页面，不显示登录用户名，如图 15.4 所示。且可以看到页面地址栏 URL 已经变为“http://localhost:8080/BBS_Struts2_1300_resultType/index.jsp”。

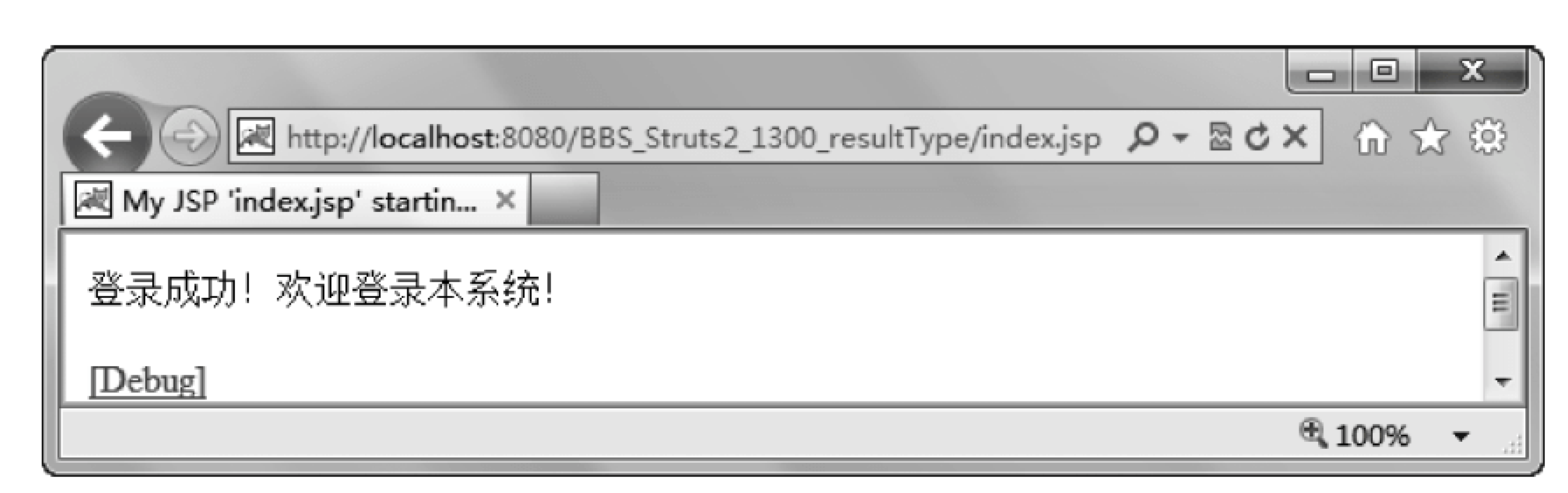


图 15.4　请求重定向到 index.jsp 页面

单击＜s:debug＞标签，打开 ValueStack，查看 Action 的实例和 Action 中封装的属性 name、password 等都已丢失，因此用户名“aaa”无法显示在页面中，如图 15.5 所示。



图 15.5　请求重定向丢失 Action 实例、属性等

3. 任务总结

dispatcher 结果类型和 redirect 结果类型的相同之处是，都是用于处理视图资源；其区别在于，dispatcher 结果类型是请求转发，redirect 结果类型是请求重定向。

15.1.4　chain 结果类型

当需要让一个 Action 处理结束后，直接将请求转发到另一个 Action 时，采用 chain 结果类型。两个 Action 就形成了 Action 链，Action 链中的 Action 共用同一值栈，数据

共享,则前一个 Action 的处理结果、请求参数和请求属性都不会丢失。

1. 同包下 Action 的转发

当前一个 Action 与后一个 Action 在同一包下时,可以直接在<result>标签中指定后一个 Action,代码如下所示。

```
<package name="user" namespace="/user" extends="struts-default">
  <action name="User_Modify_input">
    <result type="chain">User_query</result>
  </action>
  <action name="User_query" class="com.BBS.struts2.main.action.User_Action"
          method="query">
    <result>/User_modify_input.jsp</result>
  </action>
</package>
```

以上配置实现的功能是:当系统管理员提交修改用户信息的请求时,"User_Modify_input"Action 接收请求,处理后,交给"User_query"Action 处理。

再例如,当删除用户后,不显示删除用户成功界面,而返回显示用户列表界面,并显示删除成功后的用户列表。这个请求就需要两个 Action 处理,前一个 Action 处理删除用户,后一个 Action 处理显示用户列表。

其配置代码如下:

```
<package name="user" namespace="/user" extends="struts-default">
  <action name="User_delete" class="com.BBS.struts2.main.action.User_Action"
          method="delete">
    <result type="chain">User_list</result>
  </action>
  <action name="User_list" class="com.BBS.struts2.main.action.User_Action"
          method="list">
    <result>/User_list_success.jsp</result>
  </action>
</package>
```

User_delete Action 调用 User_Action 的 delete 的方法处理后,将请求转发给 User_list Action 处理,User_list Action 调用 User_Action 的 list 的方法处理后,显示 User_list_success.jsp 结果页面。

2. 不同包之间 Action 的转发

配置 chain 类型时,还可以指定如下两个参数。

(1) actionName:该参数指定请求转发的 Action 名。

(2) namespace:该参数指定请求转发的 Action 所在的命名空间。

当请求转发的两个 Action 不在同一包中时,采用以上两个参数指定后一个 Action。下面是一段使用 chain 结果类型的配置代码。

```
<package name="front" namespace="/front" extends="struts-default">
```

```
    <action name="User_add" class="com.BBS.struts2.main.action.User_Action"
            method="add">
      <result type="chain">
          <param name="actionName">User_list</param>
          <param name="namespace">main</param>
      </result>
    </action>
  </package>
  <package name="main" namespace="/main" extends="struts-default">
      <action name="User_list" class="com.BBS.struts2.main.action.User_Action"
              method="list">
        <result>/User_list_success.jsp</result>
      </action>
  </package>
```

15.1.5　redirectAction 结果类型

当需要让一个 Action 处理结束后，直接将请求重定向到另一个 Action 时，采用 redirectAction 结果类型。使用 redirectAction 结果类型时，系统将重新生成一个新请求，则前一个 Action 的处理结果、请求参数和请求属性都会丢失。

1. 同包下 Action 的重定向

当前一个 Action 与后一个 Action 在同一包下时，可以直接在<result>和<result/>标签之间指定后一个 Action，代码如下：

```
<package name="user" namespace="/user" extends="struts-default">
<action name="User_Modify_input">
    <result type="redirectAction">User_query</result>
</action>
<action name="User_query" class="com.BBS.struts2.main.action.User_Action"
        method="query">
    <result>/User_modify_input.jsp</result>
</action>
</package>
```

2. 不同包下 Action 的重定向

配置 redirectAction 类型时，还可以指定如下两个参数。

(1) actionName：该参数指定请求转发的 Action 名。

(2) namespace：该参数指定请求转发的 Action 所在的命名空间。

当请求重定向的两个 Action 不在同一包中时，采用以上两个参数指定后一个 Action。下面是一段使用 redirectAction 结果类型的配置代码。

```
<package name="front" namespace="/front" extends="struts-default">
<action name="User_add" class="com.BBS.struts2.main.action.User_Action"
        method="add">
```

```
    <result type="redirectAction">
        <param name="actionName">User_list</param>
        <param name="namespace">main</param>
    </result>
</action>
</package>
<package name="main" namespace="/main" extends="struts-default">
<action name="User_list" class="com.BBS.struts2.main.action.User_Action"
        method="list">
    <result>/User_list_success.jsp</result>
</action>
</package>
```

15.1.6 实践任务 2：比较 chain 结果类型和 redirectAction 结果类型

1. 任务说明

本任务实践 chain 结果类型和 redirectAction 结果类型的使用。

分别使用 chain 结果类型和 redirectAction 结果类型实现修改用户信息，观察两者的异同。

2. 任务实施

(1) 打开工程。打开工程 BBS_Struts2_1300_resultType。

(2) 视图层文件。

① 创建 User_modify_input.jsp，主要代码如下：

```
<body>
  修改用户
   <form action="" method="get">
       用 户 名: <input type="text"/><br>
       密    码: <input type="password" /><br>
       确认密码: <input type="password" /><br>
       <input type="submit" value="修改">
   </form>
</body>
```

② 创建 User_list_success.jsp，主要代码如下：

```
<body>
    用户列表管理页面 <br/>
    用户 1|<a href="">修改</a><br/>
    用户 2|<a href="">修改</a><br/>
    用户 3|<a href="">修改</a><br/>
</body>
```

(3) 修改 Action 类。修改 User_Action，创建查询用户的方法，代码如下：

```
package com.BBS.struts2.main.action;
import com.opensymphony.xwork2.ActionSupport;
```

```
public class User_Action extends ActionSupport{
    private String name;
    private String password;

    public String login() {
        return SUCCESS;
    }

    public String query() {
        return SUCCESS;
    }
    //属性的 getter 和 setter 方法略
    ...
}
```

(4) 配置 Action。打开配置文件 struts.xml,输入以下代码。

```
<package name="user" namespace="/user" extends="struts-default">
  <action name="User_Modify_input">
    <result type="chain">User_query</result>
  </action>
  <action name="User_qurey" class="com.BBS.struts2.main.action.User_Action"
          method="query">
    <result>/User_modify_input.jsp</result>
  </action>
</package>
```

(5) 编写视图层,提交请求,显示数据,插入 debug 标签。

① 打开 User_list_success.jsp,提交请求,代码如下:

```
<body>
    用户列表管理页面 <br/>
    用户 1|<a href="user/User_Modify_input?name=user1">修改</a><br/>
    用户 2|<a href="user/User_Modify_input?name=user2">修改</a><br/>
    用户 3|<a href="user/User_Modify_input?name=user3">修改</a><br/>
</body>
```

② 打开 User_modify_input.jsp,代码如下:

```
<body>
  修改用户
   <form action="" method="get">
       用户名: <input type="text" value="<s:property value=name'/>"/><br>
       密　码: <input type="password" /><br>
       确认密码: <input type="password" /><br>
       <input type="submit" value="修改">
   </form>
</body>
<s:debug></s:debug>
```

(6) 测试。部署工程,在浏览器的地址栏中输入 http://localhost:8080/BBS_Struts2_1300_resultType/User_list_success.jsp,打开 User_list_success.jsp 页面,如图 15.6 所示,单击"修改"超链接,打开 User_modify_input.jsp 页面,显示修改用户的用户名,如图 15.7 所示。可以看到虽然页面已经跳转到 User_modify_input.jsp,而地址栏还显示请求的 URL: http://localhost:8080/BBS_Struts2_1300_resultType/user/User_Modify_input?name=user1。



图 15.6 User_list_success.jsp 页面

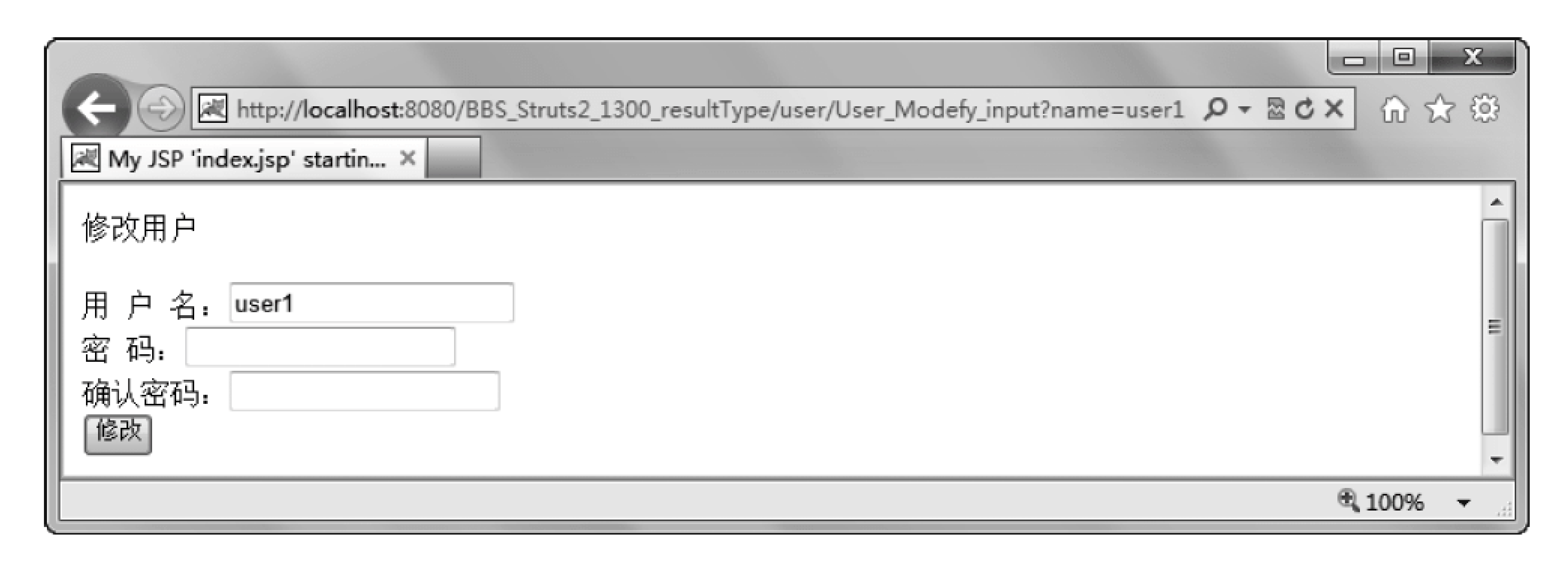


图 15.7 请求转发到 User_modify_input.jsp 页面

单击<s:debug>标签,打开 ValueStack,查看 Action 的实例和 Action 中封装的属性 name、password 等都保留,因此用户名"user1"显示在页面中,如图 15.8 所示。

因为以下配置代码未指定处理请求的 Action 类,因此创建 ActionSupport 的实例处理请求。

```
<action name="User_Modify_input">
    <result type="chain">User_query</result>
</action>
```

接着请求转发,创建 User_Action 类的实例处理请求。值栈中保存 ActionSupport 类和 User_Action 类,其请求参数、处理结果、请求属性等都保留。

且前一个 Action——ActionSupport 先压入值栈,后一个 Action——User_Action 后压入值栈。

(7) 修改 Action 配置。打开配置文件 struts.xml,修改结果类型为 redirectAction,代码如下:

```
<package name="user" namespace="/user" extends="struts-default">
  <action name="User_Modify_input">
    <result type="redirectAction">User_qurey</result>
  </action>
  <action name="User_qurey" class="com.BBS.struts2.main.action.User_Action"
```

```
            method="query">
        <result>/User_modify_input.jsp</result>
    </action>
</package>
```

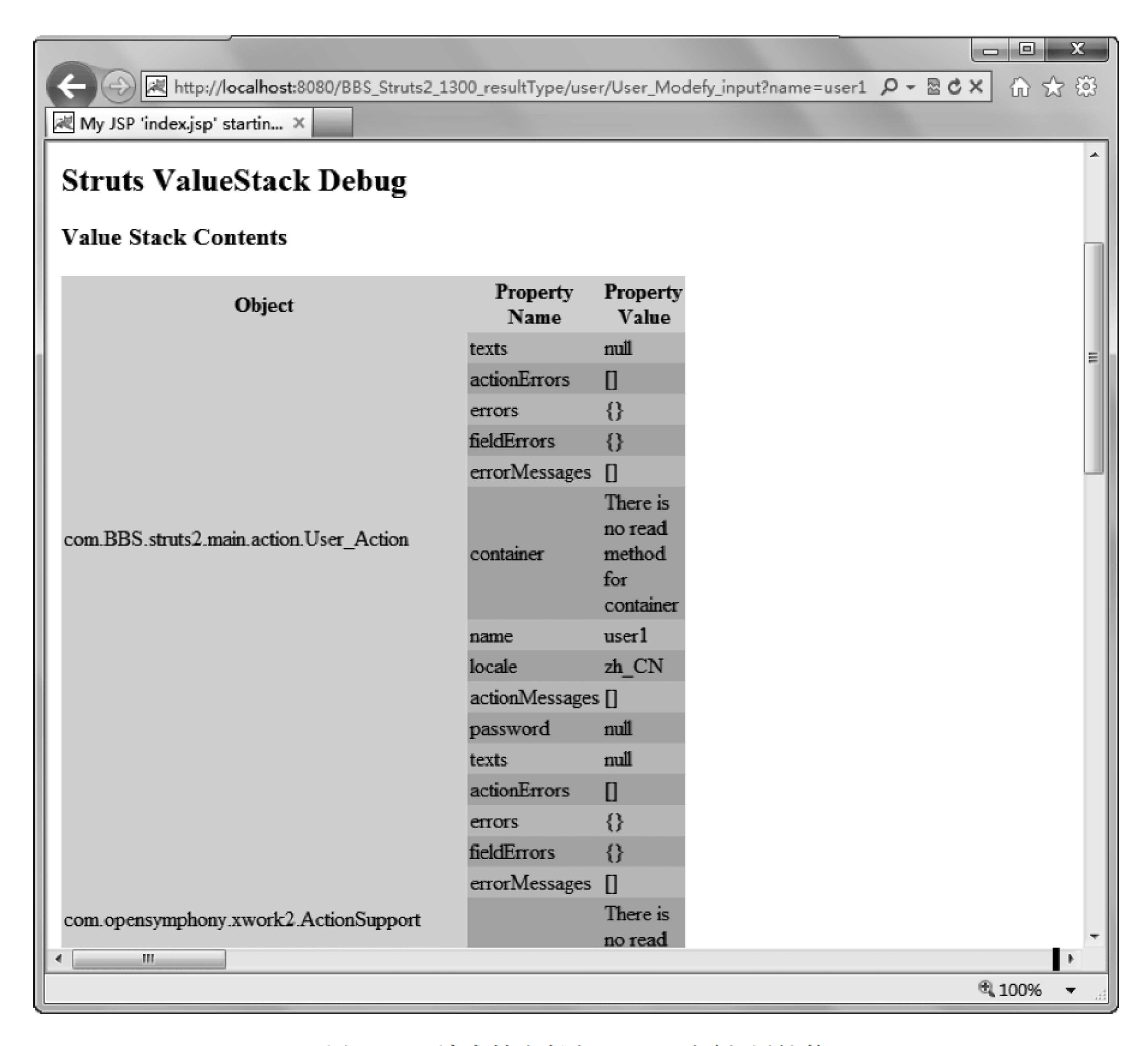


图 15.8　请求转发保留 Action 实例、属性等

(8) 测试。部署工程，在浏览器的地址栏中输入 http://localhost:8080/BBS_Struts2_1300_resultType/User_list_success.jsp，打开 User_list_success.jsp 页面，如图 15.6 所示。单击“修改”超链接，打开 User_modify_input.jsp 页面，被修改用户的用户名无法显示，如图 15.9 所示。可以看到地址栏请求的 URL 已经变为 http://localhost:8080/BBS_Struts2_1300_resultType/user/User_query.action。

单击<s:debug>标签，打开 ValueStack，查看 Action 中封装的属性 name、password 等都已丢失，因此用户名“user1”无法显示在页面中，如图 15.10 所示。

3. 任务总结

chain 结果类型和 redirectAction 结果类型的相同之处是，都用于将请求交给下一个 Action 处理；区别在于，chain 结果类型是请求转发，redirectAction 结果类型是请求重定向。

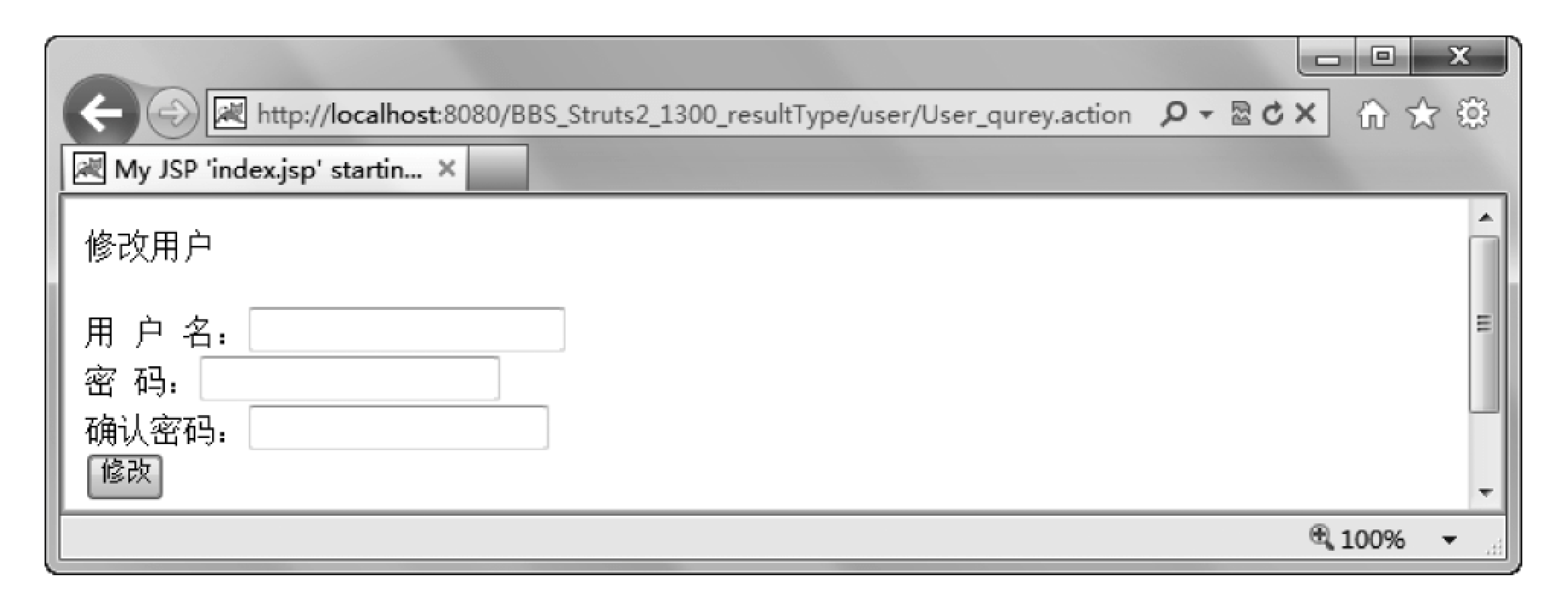


图 15.9 请求重定向到 User_modify_input.jsp 页面

Object	Property Name	Property Value
com.BBS.struts2.main.action.User_Action	texts	null
	actionErrors	[]
	errors	{}
	fieldErrors	{}
	errorMessages	[]
	container	There is no read method for container
	name	null
	locale	zh_CN
	actionMessages	[]
	password	null
com.opensymphony.xwork2.DefaultTextProvider	texts	null

图 15.10 请求重定向丢失 Action 实例、属性等

15.1.7　拓展任务：实现删除用户

1. 任务要求

实现在用户列表界面，提交删除用户的请求，调用 User_Action 的 delete 方法处理删除用户的请求，然后调用 User_Action 的 list 方法显示删除成功后的用户列表。

2. 建议步骤

（1）打开工程——BBS_Struts2_1300_resultType。

（2）修改 UserAction 类，创建 delete()方法和 list()方法。

（3）编写 struts.xml 文件，配置 Action，注意配置结果类型为 chain 或 redirectAction。

（4）修改视图层文件 User_list_success.jsp，提交删除用户的请求。

（5）部署工程。

（6）调试系统。打开浏览器，提交请求，测试程序，如有错误，进行修改。

15.2　全局结果

15.2.1　全局结果简介

Struts 2 的<result>元素也可以放在<global-results>元素中配置。当在<global-results>元素中配置<result>元素时，该<result>元素配置了一个全局结果，全局结果的作用范围是对所有的 Action 都有效。

当访问数据库失败时，系统一般跳转到数据库访问出错页面，可以将出错页面设置为全局结果集，则不需要在每个 Action 中都配置出错页面的 result 配置。例如在论坛管理系统中，当对数据库进行操作出现异常时，都返回字符串"database_error"，在配置文件 struts.xml 中配置其物理视图为 database_error.jsp。因为对用户或论坛主题等的增、删、改、查等操作都会访问数据库，都可能出现异常，在每个 Action 都配置"database_error"对应的物理视图，代码重复，而且不利于维护。可以将此配置设置为全局结果，代码如下：

```
<package name="comm" namespace="/" extends="struts-default">
    <global-results>
        <result name="database_error">/database_error.jsp</result>
    </global-results>
</package>

    <package name="user" namespace="/user" extends="comm">
        <action name="User_login" class="com.BBS.struts2.main.action.User_Action"
            method="login"><result type="redirect">/index.jsp
            </result>
        </action>
```

```
    <action name="User" class="com.BBS.struts2.main.action.User_Action">
        <result name="delete">/User_delete_success.jsp </result>
        <result name="modify">/User_modify_success.jsp</result>
        <result name="add">/User_add_success.jsp</result>
    </action>

    <action name="User_list" class="com.BBS.struts2.main.action.User_Action"
            method="list">
        <result>/User_list_success.jsp</result>
    </action>

    <action name="User_add_input">
        <result type="redirect">/User_add_input.jsp</result>
    </action>

    <action name="User_Modefy_input">
        <result type="redirectAction">User_qurey</result>
    </action>

    <action name="User_qurey" class="com.BBS.struts2.main.action.User_Action"
            method="query">
        <result>/User_modefy_input.jsp</result>
    </action>
</package>
```

将<global-results>元素配置在 name="comm"的<package>元素中，哪个<package>元素要使用此全局结果，只需要继承此<package>元素，就可以使用其全局结果。配置继承 name="comm"的<package>元素的方法是设置 extends="comm"。

提示：在 struts.xml 配置文件中，通常定义一个公共包，将一些公共配置定义在包中，任何其他包中的 Action 需要使用这些公共配置，使用 extends 属性继承公共包即可。

15.2.2 实践任务 3：配置全局结果

1. 任务说明

本任务实践配置全局结果。

在用户管理模块中，当访问数据库出现异常，均返回到 database_error.jsp 页面，如图 15.11 所示。

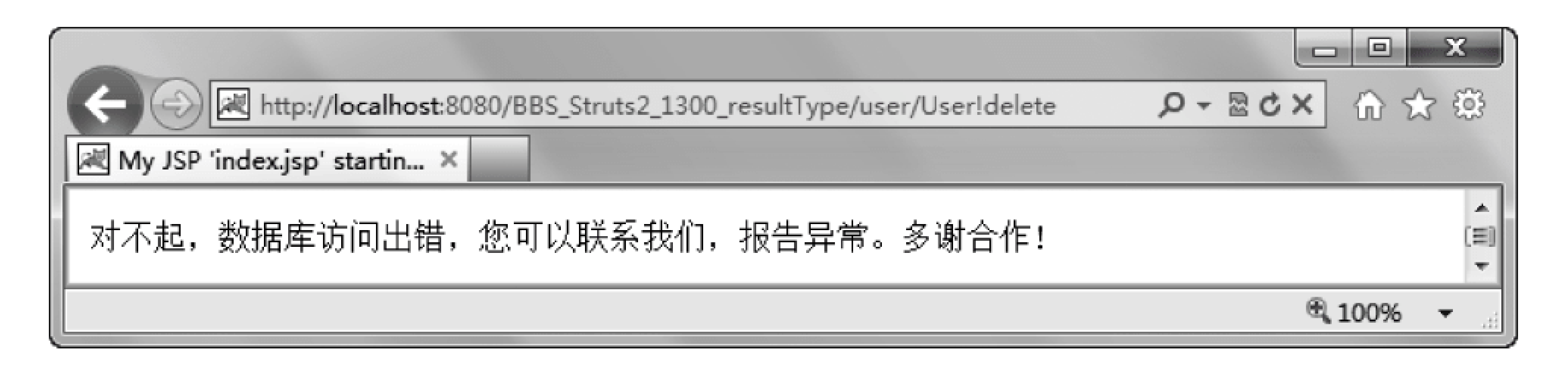


图 15.11 database_error.jsp 页面

2. 任务实施

(1) 打开工程。打开工程 BBS_Struts2_1300_resultType。

(2) 视图层文件。

① 创建 database_error.jsp,主要代码如下:

```
<body>
  对不起,数据库访问出错,您可以联系我们,报告异常。多谢合作!
</body>
```

② 创建 User_add_input.jsp,主要代码如下:

```
<body>
  修改用户
   <form action="" method="get">
       用 户 名: <input type="text"/><br>
       密    码: <input type="password" /><br>
       确认密码: <input type="password" /><br>
       <input type="submit" value="添加">
   </form>
</body>
```

③ 修改 User_list_success.jsp,主要代码如下:

```
<body>
    用户列表管理页面 <br>
    用户 1|<a href=" user/User_Modefy_input?name=user1">修改</a>|
           <a href="user/User!delete">删除</a><br>
    用户 2|<a href=" user/User_Modefy_input?name=user2">修改</a>|
           <a href="user/User!delete">删除</a><br>
    用户 3|<a href=" user/User_Modefy_input?name=user3">修改</a>|
           <a href="user/User!delete">删除</a><br>
</body>
```

(3) 修改 Action 类。修改 User_Action,创建删除用户、修改用户和添加用户的方法,并设置其返回值均为"database_error",代码如下:

```
package com.BBS.struts2.main.action;
import com.opensymphony.xwork2.ActionSupport;

public class User_Action extends ActionSupport{
    private String name;
    private String password;

    public String login() {
        return SUCCESS;
    }

    public String add() {
        return "database_error";
```

```
    }

    public String delete() {
        return "database_error";
    }

    public String modify() {
        return "database_error";
    }

    public String list() {
        return "database_error";
    }

    public String query() {
        return "database_error";
    }

    //属性的 getter 和 setter 方法略
    ...
}
```

在实际开发中，Action 类的方法根据不同的处理结果，返回不同的字符串，当访问出现异常时，返回字符串"database_error"，本任务主要练习全局结果，因此直接返回字符串"database_error"，方便测试全局结果是否配置成功。

（4）配置 Action。打开配置文件 struts.xml，输入以下代码。

```
<package name="comm" namespace="/" extends="struts-default">
    <global-results>
        <result name="database_error">/database_error.jsp</result>
    </global-results>
</package>

<package name="user" namespace="/user" extends="comm">
    <action name="User_login" class="com.BBS.struts2.main.action.User_Action"
        method="login"><result type="redirect">/index.jsp
        </result>
    </action>

    <action name="User" class="com.BBS.struts2.main.action.User_Action">
        <result name="delete">/User_delete_success.jsp </result>
        <result name="modify">/User_modify_success.jsp</result>
        <result name="add">/User_add_success.jsp</result>
    </action>

    <action name="User_list" class="com.BBS.struts2.main.action.User_Action"
        method="list">
        <result>/User_list_success.jsp</result>
    </action>
```

```
    <action name="User_add_input">
        <result type="redirect">/User_add_input.jsp</result>
    </action>

    <action name="User_Modify_input">
        <result type="redirectAction">User_query</result>
    </action>

    <action name="User_query" class="com.BBS.struts2.main.action.User_Action"
        method="query">
        <result>/User_modify_input.jsp</result>
    </action>
</package>
```

(5) 编写视图层,提交请求。

① 打开 User_list_success.jsp,提交请求,代码如下:

```
<body>
    用户列表管理页面 <br>
    用户 1|<a href="user/User_Modify_input?name=user1">修改</a>|
           <a href="user/User!delete">删除</a><br>
    用户 2|<a href="user/User_Modify_input?name=user2">修改</a>|
           <a href="user/User!delete">删除</a><br>
    用户 3|<a href="user/User_Modify_input?name=user3">修改</a>|
           <a href="user/User!delete">删除</a><br>
</body>
```

② 打开 User_modify_input.jsp,代码如下:

```
<body>
  修改用户
   <form action="user/User! modify" method="get">
       用户名: <input type="text" value="<s:property value=name'/>"/><br>
       密  码: <input type="password" /><br>
       确认密码: <input type="password" /><br>
       <input type="submit" value="修改">
   </form>
</body>
```

③ 打开 User_add_input.jsp,,代码如下:

```
<body>
    添加用户
    <form action="user/User!add" method="get">
        用户名: <input type="text" /><br>
        密  码: <input type="password" /><br>
        确认密码: <input type="password" /><br>
        <input type="submit" value="添加">
    </form>
</body>
```

(6) 测试。部署工程，在浏览器的地址栏中输入 http://localhost:8080/BBS_Struts2_1300_resultType/User_list_success.jsp，打开 User_list_success.jsp 页面，单击“修改”或“删除”超链接，浏览器打开 database_error.jsp 页面，显示数据库访问异常信息，如图 15.11 所示。

打开添加用户或修改用户界面，提交添加或修改用户请求，浏览器打开 database_error.jsp 页面，显示数据库访问异常信息，如图 15.11 所示。

3. 任务总结

全局结果简化配置，并方便配置的维护与修改。

15.2.3 实训：完善论坛管理系统的主题管理模块

实训步骤：

(1) 实现主题的增、删、改、查等操作。

(2) 根据实际需求，配置 dispatcher、redirect、chain 或 redirectAction 结果类型。

(3) 设置全局结果，配置访问数据库异常的结果页面。

(4) 测试。

15.3 本章小结

本章主要介绍了 Struts 2 的常用结果类型和全局结果的配置。本章讲解了四个常用结果类型：dispatcher、redirect、chain 和 redirectAction。

这四者的异同如下。

(1) dispatcher 和 redirect 的异同。dispatcher 和 redirect 都是将请求发到指定的 JSP 资源或 HTML 资源；但 dispatcher 是请求转发，redirect 是请求重定向。

(2) chain 和 redirectAction 的异同。chain 和 redirectAction 都是将请求结果交给 Action；但 redirectAction 处理完后重定向到一个 Action，请求参数全部丢失，Action 处理结果也全部丢失。chain 结果类型处理完后转发到一个 Action，两个 Action 共用一个值栈，请求参数、处理结果、属性等都不丢失。

本章还讲解了全局结果的配置方法和作用。

第16章

Struts 2 中的 OGNL 表达式

在 Struts 2 中，通过 OGNL 表达式可以访问值栈中的信息。OGNL 表达式功能强大，是一种可以方便地操作对象的开源表达式，在 Struts 2 中应用非常广泛。

本章要点：

- OGNL 表达式的运算规则
- OGNL 表达式访问值栈中的对象
- OGNL 表达式访问静态对象
- OGNL 表达式访问集合
- OGNL 表达式访问上下文

16.1　OGNL 简介

OGNL(Object-Graph Navigation Language，对象图导航语言)是一种功能强大的表达式语言(Expression Language，EL)，通过简单一致的表达式语法，可以任意存取对象的属性或调用对象的方法，能够遍历整个对象的结构图，实现对象属性字段的类型转化等功能。

OGNL 是 Struts 2 默认的表达式语言，是可以方便地操作对象的开源表达式语言，使用它可以在页面简单、轻松地存取数据，在 Struts 2 中应用广泛。

OGNL 相对其他表达式语言具有下面几大优势。

(1) 支持对象方法的调用，如 user.add()。

(2) 支持类静态的方法调用和值访问。

(3) 支持赋值操作和表达式串联，如表达式"price = 10，count = 8，calculatePrice()"会返回 80。

(4) 提供了对 Struts 2 应用上下文的访问机制。

(5) 可以操作集合对象。

16.2　OGNL 表达式

16.2.1　OGNL 表达式的使用方法

在 OGNL 中所有被访问者都被看做一个对象，该对象及其属性构成这个对

象的一个视图,OGNL 就是基于这个视图来获取所需的信息。OGNL 表达式将 Value Stack 看作一个“根”对象,从这个“根”对象出发,形成对象的导航视图,使用“.”对导航图进行遍历。

1. 访问 Value Stack 中的普通属性和方法

例如,某 Action 类中定义了属性 username,当提交请求时,此 Action 类的实例被压入 Value Stack,则读取 username 属性值的 OGNL 表达式就是 username。在 Action 类中定义了 add()方法,则调用 add()方法的 OGNL 表达式就是 add()。

2. 访问 Value Stack 中对象的属性和方法

如果在 Action 类中定义了 User 类的对象作为此 Action 类的属性,那么就可以访问 User 类对象的属性和调用 User 类对象的方法。

Action 类和 User 类的定义代码如下:

```
public String User_Action extends ActionSupport {
    private User user;
    public User getUser() {
        return user;
    }
    public void setUser(User user) {
        this.user = user;
    }
    ...
}
```

定义 User 类的代码如下:

```
public class User {
    private String name;
    private String password;
    private  int age;

    public String sayHello() {
        return  "User say Hello";
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }
```

```
    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }
}
```

访问 user 对象的 name 属性的 OGNL 表达式为 user. name，调用 user 对象的 sayHello()方法的 OGNL 表达式为 user. sayHello()。

3. 访问 Value Stack 中对象的属性和方法

若在 Action 类中定义 Post(帖子)类的对象作为此 Action 类的属性，Post 类的属性包括 Theme(主题)类的对象。

(1) Post_Action 类的代码如下：

```
public String Post_Action extends ActionSupport {
    private Post post;
    public Post getPost() {
        return post;
    }
    public void setPost(Post post) {
        this.post =post;
    }
    ...
}
```

(2) Post 类的代码如下：

```
public class Post{
    private String name;
    private Theme theme;                    //主题类的对象

    //置顶方法
    public String stick() {
        return "Sticky Post ";
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public Theme getTheme() {
        return theme;
```

```
    }

    public void setTheme(Theme theme) {
        this.theme = theme;
    }
}
```

(3) Theme 类的代码如下：

```
public class Theme {
    private String name;
    public Theme() {
    }
    public Theme(String name) {
        super();
        this.name = name;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    //设置版主
    public String setModerator(){
        return "Moderator is marry";
    }

    public String toString(){
        return "theme:"+name;
    }
}
```

访问 post 对象的 theme 属性的 name 属性值的 OGNL 表达式是 post.theme.name，调用 toString()方法的 OGNL 表达式是 post.theme.toString()。

OGNL 表达式处理这一过程的时候，会首先将 post 对象和 theme 对象构造成一个视图(见图 16.1)，然后根据该视图利用类的属性名逐层遍历。

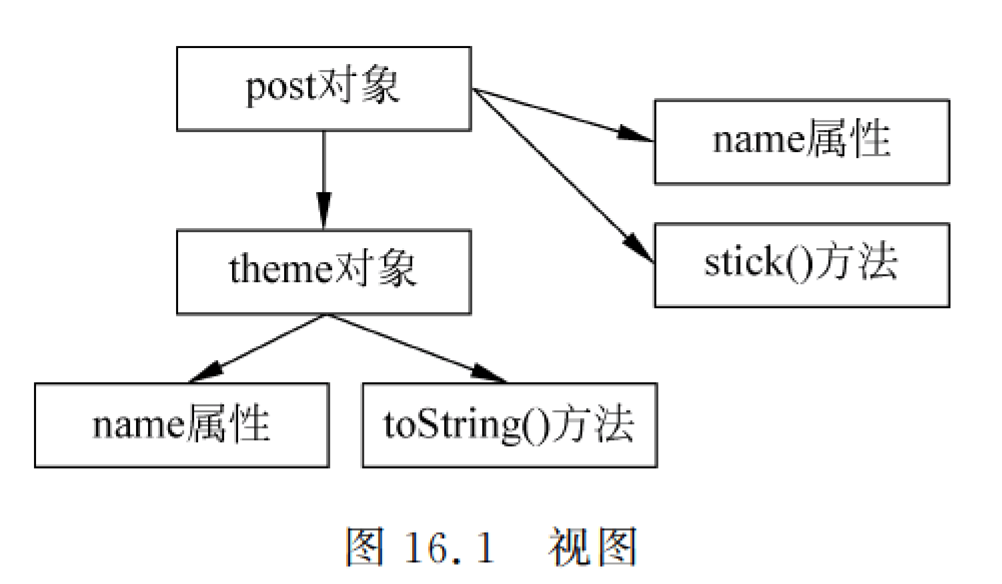


图 16.1 视图

16.2.2 实践任务 1：使用 OGNL 表达式访问 Value Stack 中的普通属性和方法

1. 任务说明

本任务实践使用 OGNL 表达式访问 Value Stack 中 Action 的属性和方法。

2. 任务实施

(1) 创建工程。创建工程 BBS_Struts2_1400_OGNL。

(2) 视图层文件。

① 创建 index.jsp,主要代码如下:

```
<body>
    访问 Action 属性和调用方法<a href="">ognl</a><br/>
</body>
```

② 创建 ognl.jsp,主要代码如下:

```
<body>
    <ol>
        <li>访问值栈中的 action 的普通属性: username = <s:property value=""/> </li>
        <li>访问值栈中 action 的普通方法: <s:property value="" /></li>
    </ol>
</body>
```

(3) 创建 Action 类。创建 OGNL_Action,代码如下:

```
package com.BBS.struts2.main.action;
import com.opensymphony.xwork2.ActionSupport;

public String User_Action extends ActionSupport {
    private String username;
    private String password;
    public String say() {
        return "hello";
    }
    //属性的 getter 和 setter 方法略
    ...
}
```

(4) 配置 Action。打开配置文件 struts.xml,输入以下代码。

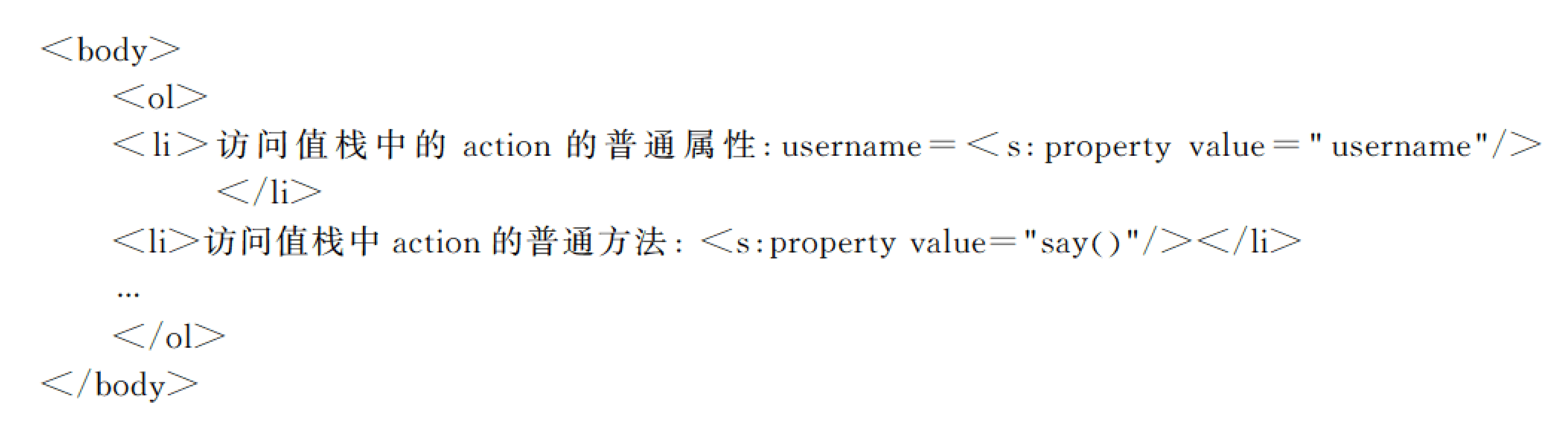
```
<package name="default" namespace="/" extends="struts-default">
  <action name="ognl" class="com.bjsxt.struts2.ognl.OgnlAction">
      <result>/ognl.jsp</result>
  </action>
</package>
```

(5) 编写视图层,提交请求,插入 OGNL 表达式。

① 打开 index.jsp,代码如下:

```
<body>
    访问 Action 属性和调用方法<a href="ognl?username=aaa">ognl</a></br>
    ...
</body>
```

② 打开 ognl.jsp，插入 OGNL 表达式，代码如下：



```
<body>
    <ol>
    <li>访问值栈中的 action 的普通属性: username=<s: property value="username"/>
        </li>
    <li>访问值栈中 action 的普通方法: <s:property value="say()"/></li>
    ...
    </ol>
</body>
```

(6) 测试。部署工程，在浏览器的地址栏中输入 http://localhost:8080/BBS_Struts2_1400_OGNL/index.jsp，打开 index.jsp 页面(见图 16.2)，单击"访问 Action 属性和调用方法"文字后的超链接，打开 ognl.jsp 页面，显示 Action 的属性 username 的值和调用 Action 的 say()方法，如图 16.3 所示。



图 16.2　index.jsp 页面 1



图 16.3　访问 Action 的普通属性及调用普通方法

3. 任务总结

本任务使用 OGNL 表达式访问 Value Stack 中的普通属性和调用普通方法。

16.2.3　实践任务 2：使用 OGNL 表达式访问 Value Stack 中对象的属性和方法

1. 任务说明

本任务实践使用 OGNL 表达式访问 Value Stack 中对象的属性和调用对象的方法。

2. 任务实施

(1) 打开工程。打开工程 BBS_Struts2_1400_OGNL。

(2) 视图层文件。

① 修改 index.jsp，主要代码如下：

```
<body>
    访问 Action 属性和调用方法<a href="ognl?username=aaa">ognl</a><br>
    访问对象的属性和调用方法<a href="">ognl</a><br>
</body>
```

② 修改 ognl.jsp，主要代码如下：

```
<body>
    <ol>
        <li>访问值栈中的 action 的普通属性: username = <s:property value="username"/>
          </li>
        <li>访问值栈中 action 的普通方法: <s:property value="say()" /></li>
        <li>访问值栈中对象的普通属性(get set 方法): <s:property value=""/> |
            <s:property value=""/> </li>
        <li>访问值栈中对象的普通方法: <s:property value=""/></li>
    </ol>
</body>
```

(3) 创建 User 类。创建 User 类，代码如下：

```
public class User {
    private String name;
    private String password;
    private  int age;

    public String sayHello() {
        return"User say Hello";
    }
    //属性的 getter 和 setter 方法略
    ...
}
```

(4) 修改 Action 类。修改 OGNL_Action，代码如下：

```
package com.BBS.struts2.main.action;
import com.opensymphony.xwork2.ActionSupport;

public String User_Action extends ActionSupport {
    private String username;
    private String password;
    private User user;

    public String say() {
        return "hello";
    }
    //属性的 getter 和 setter 方法略
    ...
}
```

(5) 配置 Action。Action 的配置代码不必修改。

(6) 编写视图层,提交请求,插入 OGNL 表达式。

① 打开 index.jsp,代码如下:

```
<body>
    访问 Action 属性和调用方法<a href="ognl?username=aaa">ognl</a><br>
    访问对象的属性和调用方法<a href="ognl?user.name=aaa&user.password=123">ognl
        </a><br>
    …
</body>
```

② 打开 ognl.jsp,插入 OGNL 表达式,代码如下:

```
<body>
    <ol>
    <li>访问值栈中的 action 的普通属性:username=<s:property value="username"/>
      </li>
    <li>访问值栈中 action 的普通方法: <s:property value="say()"/></li>
    <li>访问值栈中对象的普通属性(get set 方法): <s:property value="user.name"/> |
      <s:property value="user.password"/>  </li>
    <li>访问值栈中对象的普通方法: <s:property value="user.sayHello()"/></li>
    </ol>
</body>
```

(7) 测试。部署工程,在浏览器的地址栏中输入 http://localhost:8080/BBS_Struts2_1400_OGNL/index.jsp,打开 index.jsp 页面(见图 16.4),单击“访问对象的属性和调用方法”文字后的超链接,打开 ognl.jsp 页面,显示 user 对象的属性 name、password 的值和调用 user 对象的 sayHello()方法,如图 16.5 所示。



图 16.4 index.jsp 页面 2



图 16.5 访问对象的属性及调用对象的方法

3. 任务总结

本任务使用 OGNL 表达式访问 Value Stack 中的普通属性和调用普通方法。

16.2.4　实践任务 3：使用 OGNL 表达式访问 Value Stack 中对象的对象

1. 任务说明

在 Value Stack 中，有时某些对象中的某个属性本身也是一个对象，这个属性就像一个位于对象中的对象，本任务实践使用 OGNL 表达式访问 Value Stack 中这种对象的对象的属性和调用对象的对象的方法。

2. 任务实施

（1）打开工程。打开工程 BBS_Struts2_1400_OGNL。

（2）视图层文件。

① 修改 index.jsp，主要代码如下：

```
<body>
    访问 Action 属性和调用方法<a href="ognl?username=aaa"> ognl</a></br>
    访问对象的属性和调用方法<a href="ognl?user.name=aaa&user.password=123">ognl
      </a><br>
    访问对象的对象的属性和调用方法<a href="">ognl</a><br>
</body>
```

② 修改 ognl.jsp，主要代码如下：

```
<body>
    <ol>
        <li>访问值栈中的 action 的普通属性: username = <s:property value="username"/>
          </li>
        <li>访问值栈中 action 的普通方法: <s:property value="say()" /></li>
        <li>访问值栈中对象的普通属性(get set 方法): <s:property value="user.name"/>
          |<s:property value="user.password"/>  </li>
        <li>访问值栈中对象的普通方法: <s:property value="user.sayHello()"/></li>
        <li>访问值栈中对象的对象的普通属性(get set 方法): <s:property value=""/></li>
        <li>访问值栈中对象的对象的普通方法: <s:property value="" /></li>
    </ol>
</body>
```

（3）创建 Theme 类和 Post 类。

① Theme 类的主要代码如下：

```
public class Theme {
    private String name;
    public String getName() {
        return name;
    }
```

```
    public void setName(String name) {
        this.name = name;
    }
    //设置版主
    public String setModerator(){
        return "Moderator is marry";
    }

    public String toString(){
        return "theme:"+name;
    }
}
```

② Post 类的主要代码如下：

```
package com.bjsxt.struts2.ognl;

public class Post {
    private String name;
    private Theme theme;

    //置顶
    public String stick() {
        return "Sticky Post";
    }
    //属性的 getter 和 setter 方法略
    …
}
```

(4) 修改 Action 类。修改 OGNL_Action，代码如下：

```
package com.BBS.struts2.main.action;
import com.opensymphony.xwork2.ActionSupport;

public String User_Action extends ActionSupport {
    private String username;
    private String password;
    private Post post;

    public String say() {
        return "hello";
    }

    //属性的 getter 和 setter 方法略
    …
}
```

(5) 配置 Action。Action 的配置代码不必修改。

(6) 编写视图层，提交请求，插入 OGNL 表达式。

① 打开 index.jsp,代码如下:

```
<body>
    访问 Action 属性和调用方法<a href="ognl?username=aaa">ognl</a><br>
    访问对象的属性和调用方法<a href="ognl?user.name=aaa&user.password=123">ognl
      </a><br>
    访问对象的对象的属性和调用<a href="ognl?post.theme.name=struts2">ognl</a><br>
</body>
```

② 打开 ognl.jsp,插入 OGNL 表达式,代码如下:

```
<body>
    <ol>
    <li>访问值栈中的 action 的普通属性:username=<s:property value="username"/></li>
    <li>访问值栈中 action 的普通方法: <s:property value="say()"/></li>
    <li>访问值栈中的 action 的普通属性: username = <s:property value="username"/> </li>
    <li>访问值栈中 action 的普通方法: <s:property value="say()" /></li>
    <li>访问值栈中对象的普通属性(get set 方法): <s:property value="user.name"/> |
      <s:property value="user.password"/>   </li>
    <li>访问值栈中对象的普通方法: <s:property value="user.sayHello()"/></li>
    <li>访问值栈中对象的对象的普通属性(get set 方法):
      <s:property value="post.theme.name"/></li>
    <li>访问值栈中对象的对象的普通方法: <s:property value="post.theme.setModerator()" />
      </li>
    </ol>
</body>
```

(7) 测试。部署工程,在浏览器的地址栏中输入 http://localhost:8080/BBS_Struts2_1400_OGNL/index.jsp,打开 index.jsp 页面(见图 16.6),单击"访问对象的对象的属性和调用方法"文字后的超链接,打开 ognl.jsp 页面,显示 post 对象的 theme 对象的属性 name 的值和调用 post 对象的 theme 对象的 setModerator() 方法,如图 16.7 所示。

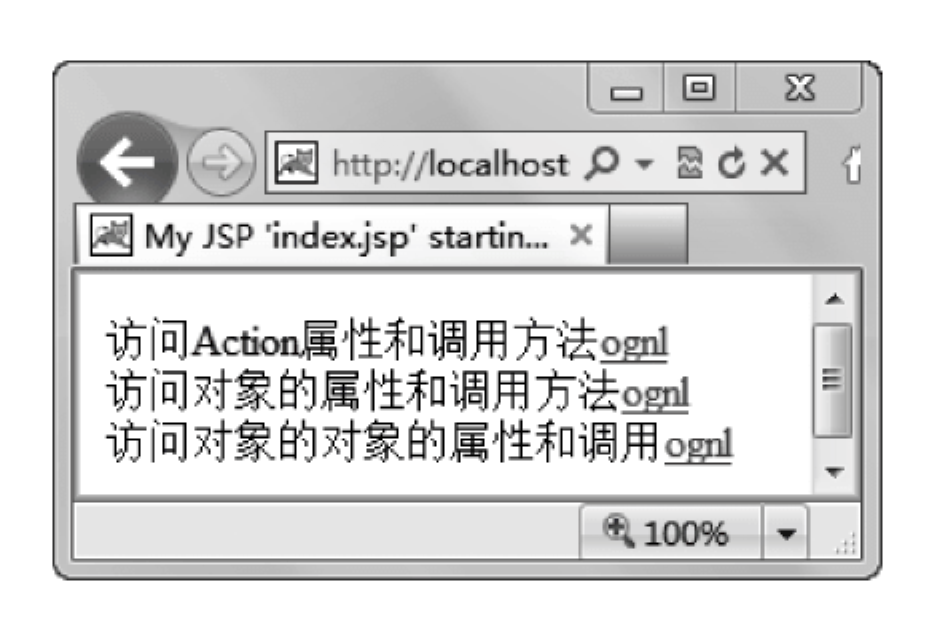


图 16.6　index.jsp 页面 3

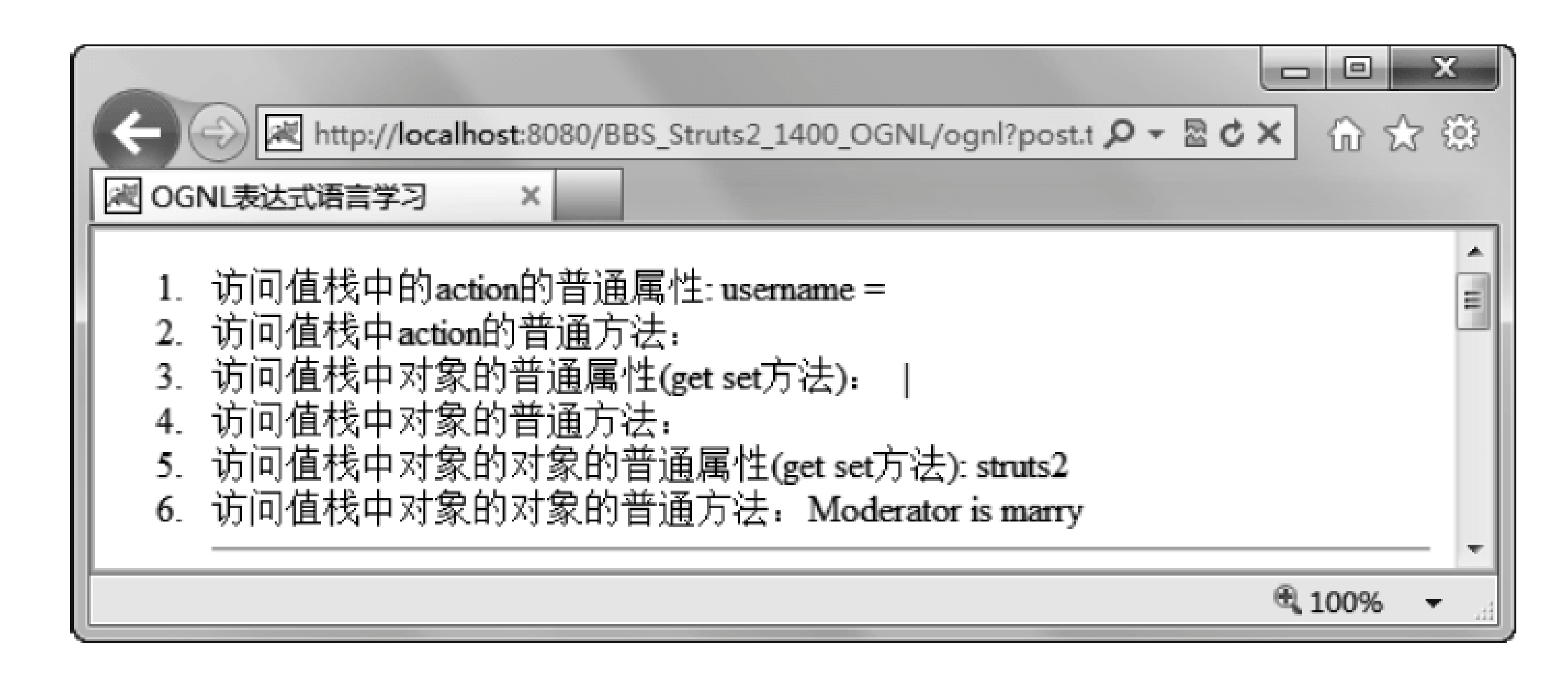


图 16.7　访问对象的对象的属性及调用对象的对象的方法

3. 任务总结

本任务使用 OGNL 表达式访问 Value Stack 中的对象的对象的属性和调用对象的对象的方法。

16.3 使用 OGNL 表达式访问静态成员

16.3.1 访问方法

OGNL 表达式还提供了访问静态成员(包括访问静态属性、调用静态方法)的方法。但需要注意的是，Struts 2 默认关闭了访问静态成员的方法，为了让 OGNL 表达式可以访问静态成员，需要在 struts.xml 中将 struts.ognl.allowStaticMethodAccess 设置为 true，代码如下：

```
<constant name="struts.ognl.allowStaticMethodAccess" value="true"/>
```

OGNL 访问类的静态属性和调用静态方法的表达式格式如下：

```
@[类全名(包括包路径)]@[方法名|属性名]
```

例如在 com.bjsxt.struts2.ognl 下定义了静态类 S，S 类中定义了静态属性 STR 和静态方法 s()，代码如下：

```
package com.bjsxt.struts2.ognl;
public class S {
    public static String STR="Static String ";

    public static String s(){
        return "Static method";
    }
}
```

访问静态类 S 的 STR 属性的 OGNL 表达式为@com.bjsxt.struts2.ognl.S@STR，调用其静态方法 s()的 OGNL 表达式为@com.bjsxt.struts2.ognl.S@s()。

如果要调用 Java 自带类的方法，表达式格式如下：

```
@@[方法名]
```

例如，调用 java.lang.Math 类的 max()方法，OGNL 表达式为@@max(参数 1，参数 2)。

16.3.2 实践任务 4：使用 OGNL 表达式访问静态类的属性和方法

1. 任务说明

本任务实践使用 OGNL 表达式访问静态类的属性和调用静态类的方法。

2. 任务实施

(1) 打开工程。打开工程 BBS_Struts2_1400_OGNL。

(2) 视图层文件。

① 修改 index.jsp,主要代码如下:

```
<body>
    访问 Action 属性和调用方法<a href="ognl?username=aaa">ognl</a><br>
    访问对象的属性和调用方法<a href="ognl?user.name=aaa&user.password=123">ognl
      </a><br>
    访问对象的对象的属性和调用方法<a href=" ognl?post.theme.name=struts2 ">ognl</a>
      <br>
    访问静态类的属性和调用方法<a href=" ">ognl</a><br/>
</body>
```

② 修改 ognl.jsp,主要代码如下:

```
<body>
    <ol>
        ...
        <hr />
        <li>访问静态方法: <s:property value=""/></li>
        <li>访问静态属性: <s:property value=" "/></li>
        <li>访问 Math 类的静态方法: <s:property value="" /></li>
        <hr />
    </ol>
</body>
```

(3) 创建 S 类。创建 S 类,主要代码如下:

```
package com.bjsxt.struts2.ognl;
public class S {
    public static String STR="Static String ";

    public static String s(){
        return "Static method";
    }
}
```

(4) Action 类。Action 类不必修改。

(5) 配置 Action。在 struts.xml 中将 struts.ognl.allowStaticMethodAccess 设置为 true,代码如下:

```
<constant name="struts.ognl.allowStaticMethodAccess" value="true"/>
<package name="default" namespace="/" extends="struts-default">
    <action name="ognl" class="com.bjsxt.struts2.ognl.OgnlAction">
        <result>/ognl.jsp</result>
    </action>
</package>
```

(6) 编写视图层,提交请求,插入 OGNL 表达式。

① 打开 index.jsp,代码如下:

```
<body>
```

```
    ...
    访问静态类的属性和调用方法<a href="ognl">ognl</a><br/>
</body>
```

② 打开 ognl.jsp，插入 OGNL 表达式，代码如下：

```
<body>
    <ol>
      ...
      <hr />
        <li>访问静态方法：<s:property value="@com.bjsxt.struts2.ognl.S@s()"/></li>
      <li>访问静态属性：<s:property value="@com.bjsxt.struts2.ognl.S@STR"/></li>
        <li>访问 Math 类的静态方法：<s:property value="@@max(5,6)" /></li>
      <hr />
    </ol>
</body>
```

(7) 测试。部署工程，在浏览器的地址栏中输入 http://localhost:8080/BBS_Struts2_1400_OGNL/index.jsp，打开 index.jsp 页面，单击“访问静态类的属性和调用方法”文字后的超链接，打开 ognl.jsp 页面，显示静态类 S 的属性 STR 的值和调用静态类 S 的 s()方法，调用 Math 类的 max()方法，如图 16.8 所示。



图 16.8 访问静态类的属性及调用静态类的方法

3. 任务总结

本任务使用 OGNL 表达式访问静态类的属性及调用静态类的方法。

16.4 使用 OGNL 表达式访问集合

在 OGNL 中，可以方便地访问 List、Map、Set 等类型的集合。

16.4.1 OGNL 对 List 的访问

OGNL 可以方便地访问 List 类型的集合，“集合名”就可以获得整个集合，“集合名

[i]"获得集合中第 i+1 个元素,"集合名.{属性名}"取得集合中每个元素的属性值,然后组合成一个集合,因此集合名.{属性名}[i]"/>和集合名[i].属性名均可以获得集合中第 i+1 个元素的属性值。

例如定义 User 类,代码如下:

```
public class User {
    private  int age;
    public User(int age){this.age=age;}
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }
    @Override
    public String toString() {
        return "user" + age;
    }
}
```

在 Action 类中,定义 User 类的集合 Users,代码如下:

```
private List<User> users = new ArrayList<User>();
public String execute() {
    users.add(new User(21));
    users.add(new User(22));
    users.add(new User(23));
}
```

则 OGNL 表达式 users 会获得集合中所有的元素,因为 User 类重写了 toString 方法,所以表达式值为{user21,user22,user23}。OGNL 表达式 users[1]访问集合中第二个元素,值为 user22;OGNL 表达式 users.{age}获得所有元素的 age 属性值所组成的集合,值为{21,22,23}。OGNL 表达式 users.{age}[1]和 users[1].age 获得的都是第二个元素的 age 属性值 22。

16.4.2　OGNL 对 Set 的访问

与访问 List 相似,OGNL 可以访问 Set 类型的集合,但是因为 Set 类型的集合没有顺序,所以只能通过"集合名"访问整个集合,无法使用"集合名[i]"的方式访问集合中的指定元素。

16.4.3　OGNL 对 Map 的访问

OGNL 可以访问 Map 类型的集合,可以使用"集合名"访问整个集合,使用"集合名.key 值"或"集合名['key 值']"访问指定元素的 value 值,使用"集合名.keys"访问集合中

所有元素的 key 值，使用“集合名. values”访问集合中所有元素的 value 值。

List、Set、Map 集合都可以通过“集合名. size()”和“集合名. size”获得集合的大小。

16.4.4 实践任务 5：使用 OGNL 表达式访问集合

1. 任务说明

本任务实践使用 OGNL 表达式访问集合。

2. 任务实施

（1）打开工程。打开工程 BBS_Struts2_1400_OGNL。

（2）视图层文件。

① 修改 index. jsp，主要代码如下：

```
<body>
    ...
    访问集合<a href=" ">ognl</a><br/>
</body>
```

② 修改 ognl. jsp，主要代码如下：

```
<body>
    <ol>
        ...
        <li>访问 List:<s:property value=""/></li>
        <li>访问 List 中某个元素:<s:property value=""/></li>
        <li>访问 List 中元素某个属性的集合:<s:property value=""/></li>
        <li>访问 List 中元素某个属性的集合中的特定值:<s:property value=""/> |
            <s:property value=""/></li>
        <li>访问 Set:<s:property value=""/></li>
        <li>访问 Set 中某个元素:<s:property value=""/></li>
        <li>访问 Map:<s:property value=""/></li>
        <li>访问 Map 中某个元素:<s:property value=""/> | <s:property value=""/> |
            <s:property value=""/></li>
        <li>访问 Map 中所有的 key:<s:property value=""/></li>
        <li>访问 Map 中所有的 value:<s:property value=""/></li>
        <li>访问容器的大小: <s:property value=""/> | <s:property value=""/> </li>
        <hr />
    </ol>
</body>
```

（3）修改 User 类、Theme 类。

① 修改 User 类，定义构造方法，在创建对象时设置属性 age 的值，并重写 toString() 方法，主要代码如下：

```
public class User {
    ...
    private  int age;
```

```
    public User(){}
    public User(int age){this.age=age;}
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }
    @Override
    public String toString() {
        return "user" + age;
    }
    ...
}
```

② 修改 Theme 类。代码如下：

```
package com.bjsxt.struts2.ognl;

public class Theme {
    private String name;
    public Theme() {
    }
    public Theme(String name) {
        super();
        this.name = name;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }

    public String setModerator(){
        return "Moderator is marry";
    }

    public String toString(){
        return "theme:"+name;
    }
}
```

(4) 修改 Action 类。修改 Action 类，创建 List、Map、Set 类型的集合并向集合中添加元素，代码如下：

```
private Set<Theme> themes = new HashSet<Theme>();

    private List<User> users = new ArrayList<User>();
```

```
    private Map<String, Theme> themeMap = new HashMap<String, Theme>();

    public String execute() {
        users.add(new User(21));
        users.add(new User(22));
        users.add(new User(23));

        themes.add(new Theme("themeA"));
        themes.add(new Theme("themeB"));
        themes.add(new Theme("themeC"));

        themeMap.put("001", new Theme("Theme 001"));
        themeMap.put("002", new Theme("Theme 002"));
        themeMap.put("003", new Theme("Theme 003"));

        return SUCCESS;
    }
```

(5) 配置 Action。Action 的配置代码不需要修改。

(6) 编写视图层,提交请求,插入 OGNL 表达式。

① 打开 index.jsp,代码如下:

```
<body>
    ...
    访问集合<a href="ognl">ognl</a><br/>
</body>
```

② 打开 ognl.jsp,插入 OGNL 表达式,代码如下:

```
<body>
    <ol>
        ...
        <li>访问 List:<s:property value="users"/></li>
        <li>访问 List 中某个元素:<s:property value="users[1]"/></li>
        <li>访问 List 中元素某个属性的集合:<s:property value="users.{age}"/></li>
        <li>访问 List 中元素某个属性的集合中的特定值:
            <s:property value="users.{age}[0]"/> |
            <s:property value="users[1].age"/></li>
        <li>访问 Set:<s:property value="themes"/></li>
        <li>访问 Set 中某个元素:<s:property value="themes[1]"/></li>
        <li>访问 Map:<s:property value="themeMap"/></li>
        <li>访问 Map 中某个元素:<s:property value="themeMap.001"/> |
                                <s:property value="themeMap['001']"/> |
                                <s:property value="themeMap[\"001\"]"/></li>
        <li>访问 Map 中所有的 key:<s:property value="themeMap.keys"/></li>
        <li>访问 Map 中所有的 value:<s:property value="themeMap.values"/></li>
        <li>访问容器的大小: <s:property value="themeMap.size()"/> |
                            <s:property value="users.size"/> </li>
        <hr/>
    </ol>
</body>
```

（7）测试。

部署工程，在浏览器的地址栏中输入 http://localhost:8080/BBS_Struts2_1400_OGNL/index.jsp，打开 index.jsp 页面，单击"访问集合"文字后的超链接，打开 ognl.jsp 页面，显示集合的访问结果，如图 16.9 所示。

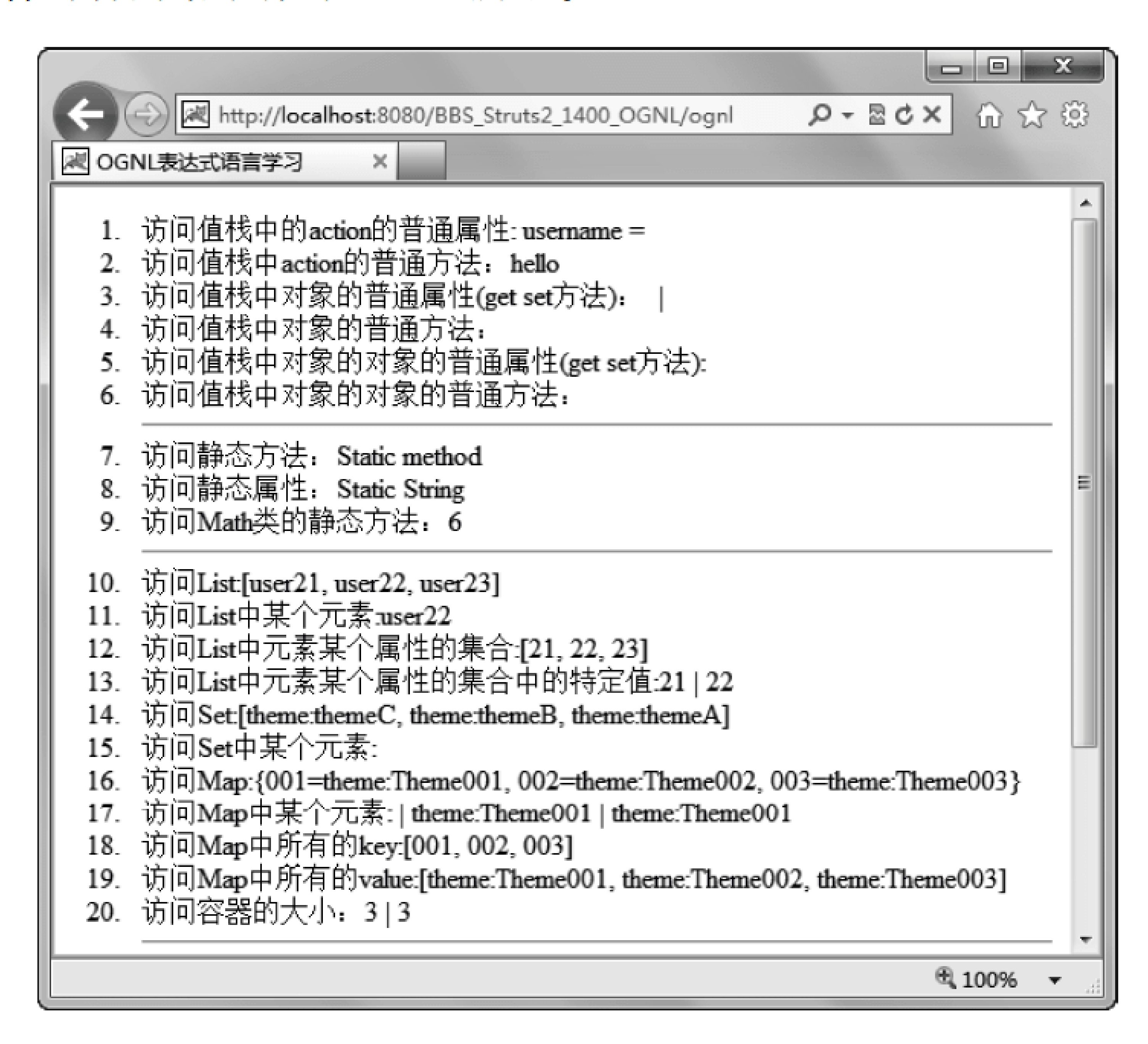


图 16.9　访问集合

3. 任务总结

本任务使用 OGNL 表达式访问集合。

本任务在 Action 类中分别创建了 List 类型的集合 users、Set 类型的集合 themes、Map 类型的集合 themeMap，在每个集合中插入三个元素。

OGNL 表达式 users 访问整个 List 集合，因为 User 类重写了 toString 方法，所以在页面显示[user21，user22，user23]；OGNL 表达式 users[1]访问 List 集合中第二个元素，在页面显示 user22；OGNL 表达式 users.{age}访问 List 集合中所有元素的 age 属性所组成的集合，在页面显示[21，22，23]；OGNL 表达式 users.{age}[0]访问 List 集合中所有元素的 age 属性所组成的集合的第一个元素，在页面显示 21；OGNL 表达式 users[1].age 访问 List 集合中第二个元素的 age 属性值，在页面显示 22。

OGNL 表达式 themes 访问整个 Set 集合，因为 Theme 类重写了 toString 方法，所以在页面显示[theme:themeC，theme:themeB，theme:themeA]；因为 Set 集合中的元素是无序的，所以 OGNL 表达式 themes[1]不能访问 Seb 集合中任何一个元素，因此在页面无显示。

OGNL 表达式 themeMap 访问整个 Map 集合，因为 Theme 类重写了 toString 方法，所以在页面显示{001 = theme: Theme001，002 = theme: Theme002，003 = theme:

Theme003}；OGNL 表达式 themeMap.001、themeMap['001']和 themeMap[\"001\"]访问 Map 集合中 key 值为 001 的元素，在页面显示此元素的 value 值 theme：Theme001；OGNL 表达式 themeMap.keys 访问 Map 集合中所有元素的 key 值，在页面显示 key 值所组成的集合[001，002，003]；OGNL 表达式 themeMap.values 访问 Map 集合中所有元素的 value 值，在页面显示 value 值所组成的集合[theme：Theme001，theme：Theme002，theme：Theme003]。

OGNL 表达式 themeMap.size()和 users.size 获得集合的大小，在页面显示集合中元素的个数 3。

16.5 使用 OGNL 表达式访问 Stack Context

16.5.1 访问方法

使用 OGNL 中可以方便地访问 Stack Context 栈上下文。Struts 2 的栈上下文也是由多个对象组成的，OGNL 通过“#”对栈上下文进行访问。

由于栈上下文中存储对象的作用域不同，在不同的作用域中可能包含相同的属性，这种情况下就可能需要手动为 OGNL 指定作用域。假设在不同作用域中都包含了 id 属性，那么通过 OGNL 表达式来获得 id 对应值的表达式如表 16.1 所示。

表 16.1 获得对应作用域中 id 值的 OGNL 表达式

名　称	作　　用	示　　例
parameters	读取请求参数	<s:property value="#parameters.id"/>
request	读取 request 中的属性	<s:property value="#request.id"/>
session	读取 session 中的属性	<s:property value="#session.id"/>
application	读取 application 中的属性	<s:property value="#application.id"/>
attr	读取前面 4 种作用域的属性以 parameters→request→session→application 顺序访问	<s:property value="#attr.id"/>

这里比较特殊的作用域是 attr，当通过#attr.id 来访问时，系统会按照作用域从小到大依次遍历，如图 16.10 所示。

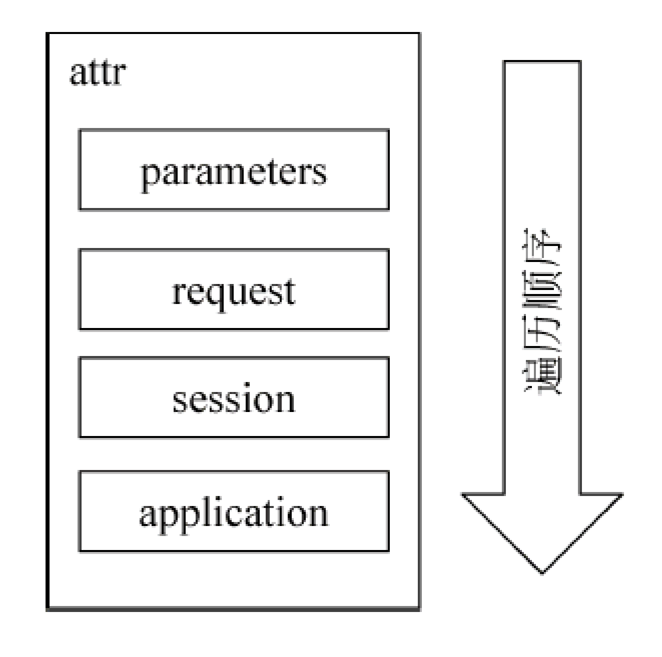


图 16.10　attr 访问时的遍历顺序

attr 虽然可以访问四个作用域，但也因此降低了其可读性，所以并不建议使用。

16.5.2　实践任务 6：使用 OGNL 表达式访问 Stack Context 中的值

1. 任务说明

本任务实践使用 OGNL 表达式访问 Stack Context 中的值。

2. 任务实施

（1）打开工程。打开工程 BBS_Struts2_1400_OGNL。

（2）视图层文件。

① 修改 index.jsp，主要代码如下：

```
<body>
    ...
    访问上下文<a href="">ognl</a><br/>
</body>
```

② 创建 ognlContext.jsp，主要代码如下：

```
<body>
    访问 parameters:<s:property value=""/><br />
    访问 request: <s:property value=""/> <br />
    访问 session: <s:property value=""/><br />
    访问 application: <s:property value=""/> <br />
    访问 attr: <s:property value=""/>
    <s:debug></s:debug>
</body>
```

（3）创建 Action 类。创建 OgnlContextAction 类，向 request、session 和 application 添加属性，代码如下：

```
import java.util.Map;
import com.opensymphony.xwork2.ActionContext;
import com.opensymphony.xwork2.ActionSupport;

public class OgnlContextAction extends ActionSupport {

    private Map request;
    private Map session;
    private Map application;

    public OgnlContextAction() {
        ActionContext actionContext = ActionContext.getContext();
        request = (Map) actionContext.get("request");
        session = actionContext.getSession();
```

```
            application = actionContext.getApplication();
        }

        public String execute() {
            request.put("key_r1", "value_r1");
            session.put("key_s1", "value_s1");
            application.put("key_a1", "value_a1");
            return "success";
        }
    }
```

(4) 配置 Action。代码如下:

```
<package name="default"namespace="/"extends="struts-default">
    ...
    <action name="ognlContext"class="com.bjsxt.struts2.ognl.OgnlContextAction">
            <result>/ognlContext.jsp</result>
    </action>
</package>
```

(5) 编写视图层,提交请求,插入 OGNL 表达式。

① 打开 index.jsp,代码如下:

```
<body>
    ...
    访问上下文<a href="ognlContext?key_p1=value_p1">ognl</a><br/>
</body>
```

② 打开 ognlContext.jsp,代码如下:

```
<body>
    访问 parameters:<s:property value="#parameters.key_p1"/><br />
    访问 request: <s:property value="#request.key_r1"/> <br />
    访问 session: <s:property value="#session.key_s1"/><br />
    访问 application: <s:property value="#application.key_a1"/> <br />
    访问 attr: <s:property value="#attr.key_a1"/>
    <s:debug></s:debug>
</body>
```

(6) 测试。部署工程,在浏览器的地址栏中输入 http://localhost:8080/BBS_Struts2_1400_OGNL/index.jsp,打开 index.jsp 页面,单击"访问上下文"文字后的超链接,打开 ognlContext.jsp 页面,显示上下文的访问结果,如图 16.11 所示。

3. 任务总结

本任务使用 OGNL 表达式访问 Stack Context 中的值,访问 Stack Context 中的值需在 OGNL 表达式前加"#"号,并指明作用域。

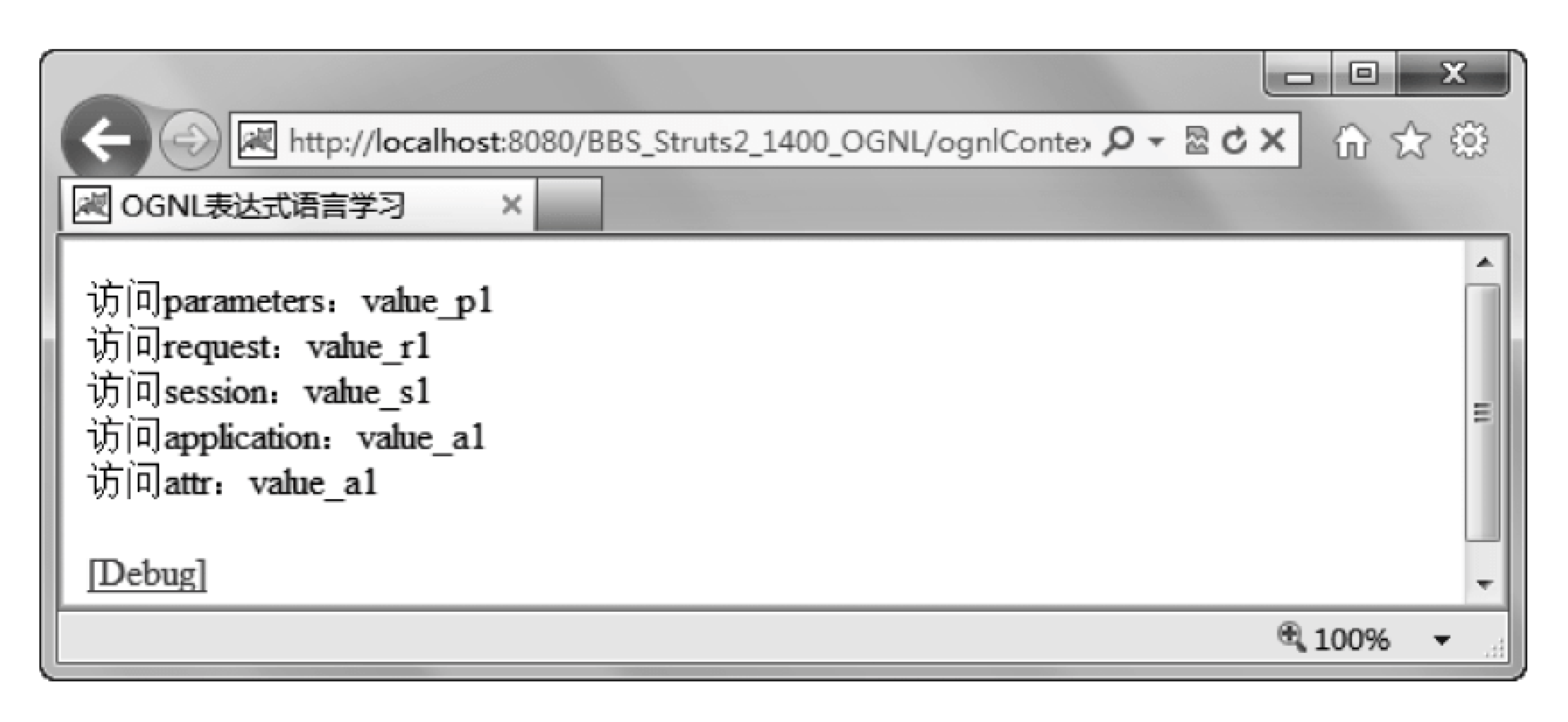


图 16.11　访问上下文

16.5.3　实训：完善论坛管理系统的用户管理模块

实训步骤：

（1）实现用户的增、删、改、查等操作。

（2）根据实际需求，在页面使用 OGNL 表达式显示数据。

在用户信息列表页面输出用户的个数和前两个用户的信息。

（3）测试。

16.6　本章小结

本章主要介绍了 OGNL 表达式，讲解了 OGNL 表达式的语法规则，示例讲解了如何使用 OGNL 表达式访问 ValueStack 中的属性和调用 ValueStack 中的方法，如何访问静态元素、如何访问集合、如何访问 Stack Context 中的值。

第17章 Struts 2 的标签库

Struts 2 作为一个 MVC 框架，提供了大量的标签供开发人员使用，包括数据标签、控制标签、表单标签、非表单 UI 标签及 Ajax 标签。Struts 标签不仅实现了 HTML 标签的功能，更提供了一些特殊功能，大大简化了数据输入输出、页面流程控制和页面效果生成等视图层代码的编写。在实际开发中，灵活地使用 Struts 2 标签将会大大提高页面开发的效率。

本章要点：

- 使用标签库的方法
- 常用数据标签
- 常用控制标签
- 常用表单标签
- 常用非表单标签

17.1 Struts 2 标签简介

在开始学习使用 Struts 2 标签之前，需要先对 Struts 2 标签有一个初步的了解。

17.1.1 标签的优势

在标签没有诞生前，Web 应用中页面与用户交互的功能，需要通过在 JSP 页面中添加 Java 代码来实现。这样做的结果是，导致大量的 Java 代码与 HTML 标签掺杂在一起，给开发和维护都带来了很大的阻碍。

应用 Struts 2 标签以后，页面中不再出现 Java 代码，取而代之的是各种 Struts 2 标签，从而大大地减少了 JSP 文件的代码，整个文件也显得更加整洁、条理清晰，而且 Struts 2 标签的一些特殊功能，使开发和维护工作都变得十分容易，这对于实际开发来说是非常有意义的。

17.1.2 初步认识 Struts 2 标签

Struts 2 标签皆以“s:”开头，比如<s:property>、<s:debug>、<s:date>等。应用 Struts 2 标签也非常容易，只需要在页面中添加<%@ taglib uri=

"/struts-tags"prefix="s"%>声明即可。

标签声明中的 prefix 参数指定标签库的前缀，通过前缀关联，当遇到<s:...>类型的标签时，系统就从 Struts 2 标签库中寻找这个标签，uri 参数指定标签库所在的位置。

Struts 2 官方 API 将标签分为两大类：通用标签和 UI 标签。其中，通用标签又分为数据标签和控制标签两种标签，UI 标签又分为表单标签、非表单 UI 标签和 Ajax 标签三种标签，各种标签的作用表述如下。

(1) 数据标签：主要用于从值栈中取值或将变量、对象等存入值栈中。

(2) 控制标签：主要用于实现分支、循环等流程控制。

(3) 表单标签：主要用于生成 HTML 页面的 form 元素以及普通表单元素。

(4) 非表单 UI 标签：主要用于生成非表单元素的可视化元素。

(5) Ajax 标签：用于支持 Ajax(Asynchronous JavaScript and XML)效果。

17.2　数据标签

数据标签用于对数据进行相关操作，通常是从值栈中读取指定的数据或将数据存入值栈中，常用的数据标签包含以下几个。

(1) property：该标签用于输出值栈中的某个值。

(2) debug：该标签用于在页面上生成一个调试链接，当单击该链接时，可以看到当前 ValueStack 和 Stack Context 中的内容。

(3) bean：该标签用于创建一个 JavaBean 实例。如果指定了 var 属性，则创建的 JavaBean 实例将放入 StackContext 中。

(4) param：该标签通常作为 bean 等标签的子标签，用于为父级标签设置一个参数。

(5) set：该标签用于在指定范围内设置一个变量。

(6) date：该标签用于格式化输出一个日期。

17.2.1　property 标签

property 标签的作用是输出值栈中指定的值。

property 标签的常用属性有三个，各属性说明如表 17.1 所示。

表 17.1　property 标签属性及说明

属性名	是否必需	默认值	类型	说明
default	false		String	用于设置默认的输出值 如果 value 属性指定输出的值为 null，则输出此默认值
escape	false	true	Boolean	用于设置要输出的内容是否要经过 HTML 的转义，默认为 true，将 HTML 标签解析为 OGNL 表达式。如果设置为 false，则不进行解析，作为 HTML 标签输出
value	false	<top of stack> 栈顶的值	Object	用于指定要输出的值，如果没有指定 value 属性，则默认输出 ValueStack 栈顶的值

property 标签 value 属性值为 OGNL 表达式，根据 OGNL 表达式计算结果可以输出 ValueStack 和 StackContext 中的值。

下面是一些示例。

（1）输出 ValueStack 中的值。

```
<s:property value="username"/>
```

说明：显示结果为 username 的值。

（2）输出字符串常量。

```
<s:property value="'username'"/>
```

说明：显示结果为字符串“username”，value 属性值默认为 OGNL 表达式，因此要输出字符串，需要加单引号。

（3）输出默认值。

```
<s:property value="admin" default="管理员"/>
```

说明：当 admin 的值为 null 时，输出 default 属性所指定的默认值“管理员”。

（4）输出 HTML 标签。

```
<s:property value="'<hr/>'" escape="false"/>
```

说明：当不设置 escape 属性或设置 escape 属性值为 true 时，输出字符串“<hr/>”，当设置 escape 属性值为 false 时，将字符串“<hr/>”解析为 HTML 标签，输出一条横线。

（5）输出 StackContext 中的值。

```
<s:property value="#session.username"/>
```

说明：显示结果为 session 中 username 的值。

注意：输出 StackContext 中的值需要在 OGNL 表达式前加“#”号。

17.2.2 debug 标签

debug 标签主要用于在页面上进行辅助调试，它会在页面上生成一个超链接，单击该链接可以查看 ValueStack 和 StackContext 中的所有信息。

我们在前面的章节中已经使用过 debug 标签，关于此标签的用法不再赘述。下面讲解一下该标签的作用。

在 Struts 应用中每提交一个请求，会创建一个 Action 实例来处理请求，并同时创建一个值栈，保存此 Action 实例的所有上下文信息。因此在开发系统时，可以通过 debug 标签查看 Action 实例的上下文信息（包括 ValueStack 和 StackContext），调试程序是否按预期创建实例、传递参数等。

同时，当页面需要存取 ValueStack 和 StackContext 中的数据时，可以使用 debug 标签查看对象名、属性名等信息，以辅助代码的编写。

总之，debug 标签为开发和调试系统提供了便捷地查看信息的方法，提高了开发和维护系统的效率。

17.2.3　实践任务 1：使用 property 标签和 debug 标签

1. 任务说明

本任务实践使用 property 标签输出指定的值和使用 debug 标签帮助开发、调试程序。

2. 任务实施

（1）创建工程。创建工程 BBS_Struts2_1500_tags。

（2）视图层文件。

① 创建 login.jsp，主要代码如下：

```
<body>
    Struts2 Tags
        <form name="f" action="login/login1" method="post">
            用户名:
            <input type="text" name="username" />
            密码:
            <input type="text" name="password" />
            <br />
            <input type="submit" value="登录 1" />
        </form>
</body>
```

② 创建 index.jsp，主要代码如下：

```
<body>
    <ol>
        <li>property ONGL 表达式:</li>
        <li>property 取值为字符串:</li>----加上单引号,是字符串
        <li>property 设定默认值: </li>-----取不到值,则显示默认值
        <li>roperty 设定 HTML: </li>不将 HTML 标签翻译成字符串,默认是 escape=
            "true",会解析 HTML 标签为普通字符串.
        <li>property 取 session 里的值:</li>
        <debug></debug>
    </ol>
</body>
```

（3）创建 Action 类。创建 LoginAction，代码如下：

```
package com.BBS.struts2.main.action;
import com.opensymphony.xwork2.ActionSupport;
import java.util.Map;
import org.apache.struts2.interceptor.SessionAware;
```

```
public String LoginAction extends ActionSupport  implements SessionAware{
    private String username;
    private String password;
    private Map<String, Object> session;
    public String execute() {
        session.put("user_name", username);
        return SUCCESS;
    }
    @Override
    public void setSession(Map<String, Object> session) {
        this.session=session;
    }
    //属性的 getter 和 setter 方法略
    ...
}
```

(4) 配置 Action。打开配置文件 struts.xml,输入以下代码。

```
<package name="login" extends="struts-default" namespace="/">
    <action name="login" class="com.cvit.struts2.user.action.LoginAction">
        <result>/index.jsp</result>
    </action>
</package>
```

(5) 编写视图层,插入 property 标签。打开 index.jsp,插入代码如下:

```
<body>
    <ol>
        <li>property ONGL 表达式: <s:property value="username" /></li>
        <li>property 取值为字符串: <s:property value="'username'" /></li>
        ----加上单引号,是字符串
        <li>property 设定默认值:
            <s:property value="admin" default="管理员" />
        </li>
        -----取不到值,则显示默认值
        <li>
            property 设定 HTML:
            <s:property value="'<hr/>'" escape="false" />
        </li>
        不将 HTML 标签翻译成字符串,默认是 escape="true",会解析 HTML 标签为普通字符串。
        <li>
            property 取 session 里的值:
            <s:property value="#session.username" />
        </li>
        <debug></debug>
    </ol>
</body>
```

(6) 测试。部署工程，在浏览器的地址栏中输入 http://localhost:8080/BBS_Struts2_1500_tags/login.jsp，打开 login.jsp 页面，输入用户名为“aaa”、密码为“123”(见图 17.1)，单击“登录 1”按钮，打开 index.jsp 页面，显示数据，如图 17.2 所示。

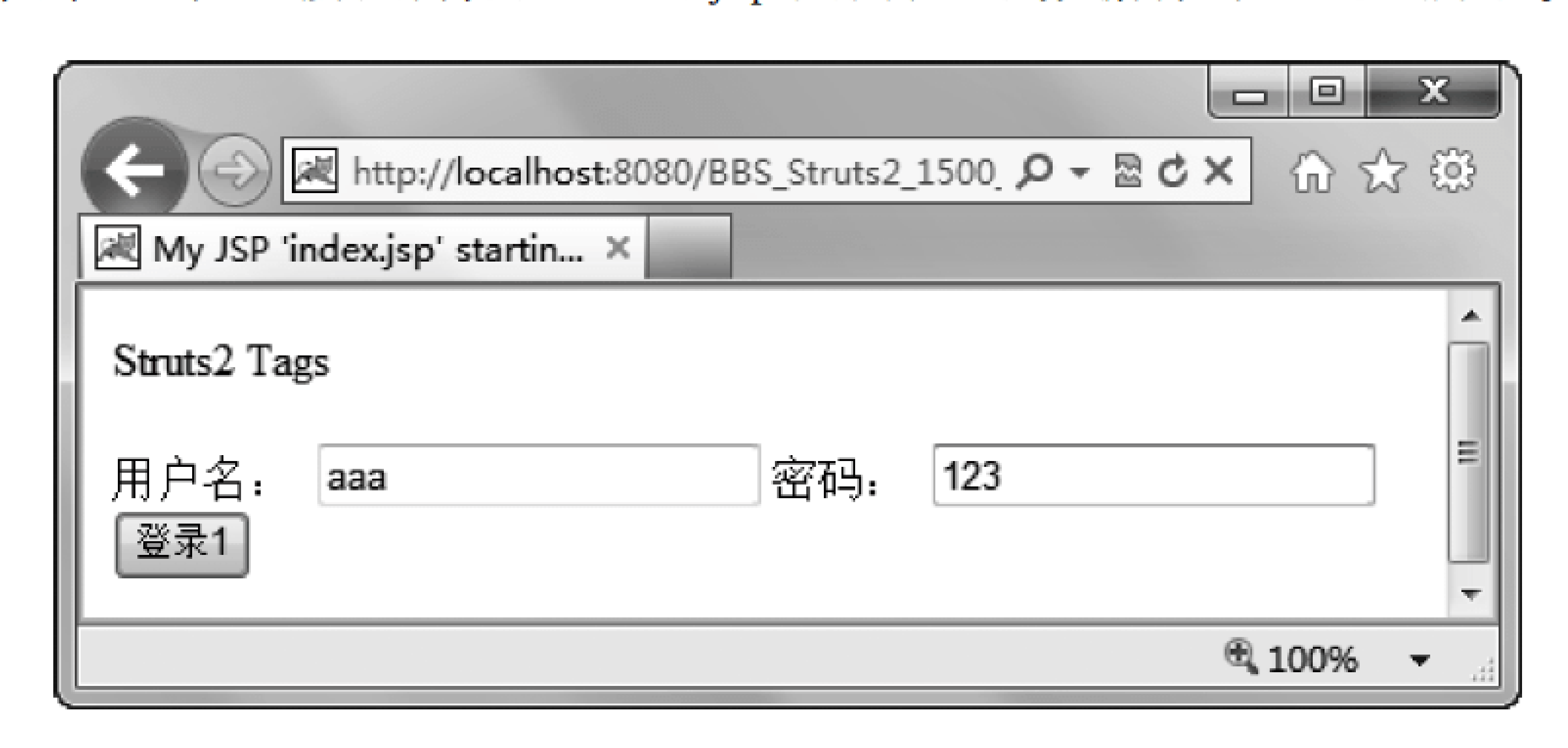


图 17.1　login.jsp 页面



图 17.2　index.jsp 页面 1

第 1 行<s:property value="username"/>计算 OGNL 表达式 username 的值，输出字符串"aaa"。

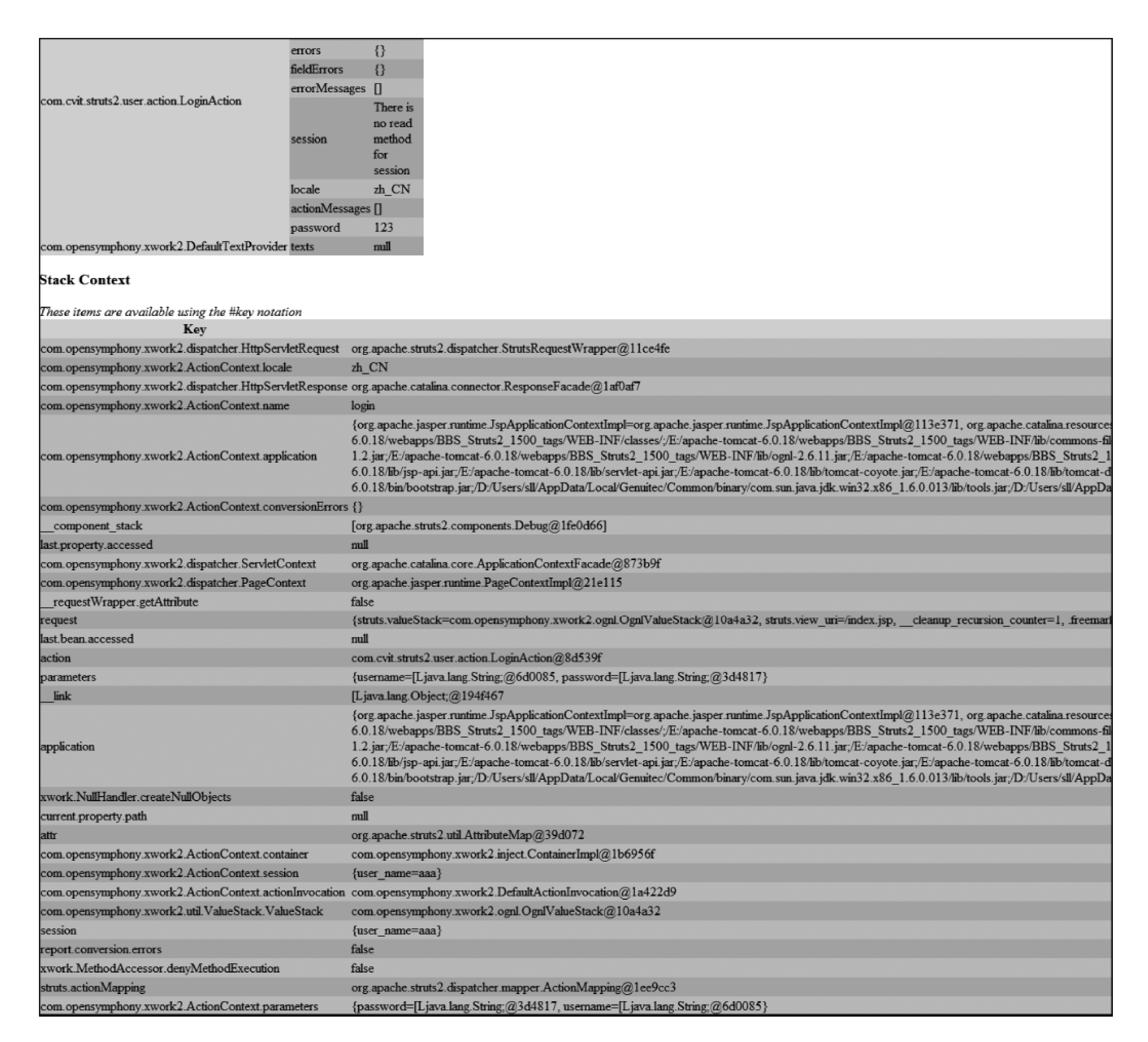
第 2 行<s:property value="'username'"/>输出字符串"username"。

第 3 行<s:property value="admin" default="管理员" />输出字符串"管理员"，因为 admin 的值为空，所以输出 default 属性指定的默认值。

第 4 行<s:property value="'<hr/>'" escape="false" />输出一条横线，因为 escape="false"指定不解析'<hr/>'为字符串，按 HTML 标签输出，<hr/>标签在页面显示一条横线。

第 5 行<s:property value="#session.username"/>输出 session 中 username 的值——字符串"aaa"。

单击 Debug 超链接，查看 ValueStack 和 StackContext 中的所有信息，可以看到 ValueStack 中 username 的值和 session 中 username 的值，如图 17.3 所示。



com.cvit.struts2.user.action.LoginAction	errors	{}
	fieldErrors	{}
	errorMessages	[]
	session	There is no read method for session
	locale	zh_CN
	actionMessages	[]
	password	123
com.opensymphony.xwork2.DefaultTextProvider	texts	null

Stack Context

These items are available using the #key notation

Key	
com.opensymphony.xwork2.dispatcher.HttpServletRequest	org.apache.struts2.dispatcher.StrutsRequestWrapper@11ce4fe
com.opensymphony.xwork2.ActionContext.locale	zh_CN
com.opensymphony.xwork2.dispatcher.HttpServletResponse	org.apache.catalina.connector.ResponseFacade@1af0af7
com.opensymphony.xwork2.ActionContext.name	login
com.opensymphony.xwork2.ActionContext.application	{org.apache.jasper.runtime.JspApplicationContextImpl=org.apache.jasper.runtime.JspApplicationContextImpl@113e371, org.apache.catalina.resource 6.0.18/webapps/BBS_Struts2_1500_tags/WEB-INF/classes/;/E:/apache-tomcat-6.0.18/webapps/BBS_Struts2_1500_tags/WEB-INF/lib/commons-fi 1.2.jar;/E:/apache-tomcat-6.0.18/webapps/BBS_Struts2_1500_tags/WEB-INF/lib/ognl-2.6.11.jar;/E:/apache-tomcat-6.0.18/webapps/BBS_Struts2_1 6.0.18/lib/jsp-api.jar;/E:/apache-tomcat-6.0.18/lib/servlet-api.jar;/E:/apache-tomcat-6.0.18/lib/tomcat-coyote.jar;/E:/apache-tomcat-6.0.18/lib/tomcat-d 6.0.18/bin/bootstrap.jar;/D:/Users/sll/AppData/Local/Genuitec/Common/binary/com.sun.java.jdk.win32.x86_1.6.0.013/lib/tools.jar;/D:/Users/sll/AppDa
com.opensymphony.xwork2.ActionContext.conversionErrors	{}
__component_stack	[org.apache.struts2.components.Debug@1fe0d66]
last.property.accessed	null
com.opensymphony.xwork2.dispatcher.ServletContext	org.apache.catalina.core.ApplicationContextFacade@873b9f
com.opensymphony.xwork2.dispatcher.PageContext	org.apache.jasper.runtime.PageContextImpl@21e115
__requestWrapper.getAttribute	false
request	{struts.valueStack=com.opensymphony.xwork2.ognl.OgnlValueStack@10a4a32, struts.view_uri=/index.jsp, __cleanup_recursion_counter=1, .freemar
last.bean.accessed	null
action	com.cvit.struts2.user.action.LoginAction@8d539f
parameters	{username=[Ljava.lang.String;@6d0085, password=[Ljava.lang.String;@3d4817}
__link	[Ljava.lang.Object;@194f467
application	{org.apache.jasper.runtime.JspApplicationContextImpl=org.apache.jasper.runtime.JspApplicationContextImpl@113e371, org.apache.catalina.resource 6.0.18/webapps/BBS_Struts2_1500_tags/WEB-INF/classes/;/E:/apache-tomcat-6.0.18/webapps/BBS_Struts2_1500_tags/WEB-INF/lib/commons-fi 1.2.jar;/E:/apache-tomcat-6.0.18/webapps/BBS_Struts2_1500_tags/WEB-INF/lib/ognl-2.6.11.jar;/E:/apache-tomcat-6.0.18/webapps/BBS_Struts2_1 6.0.18/lib/jsp-api.jar;/E:/apache-tomcat-6.0.18/lib/servlet-api.jar;/E:/apache-tomcat-6.0.18/lib/tomcat-coyote.jar;/E:/apache-tomcat-6.0.18/lib/tomcat-d 6.0.18/bin/bootstrap.jar;/D:/Users/sll/AppData/Local/Genuitec/Common/binary/com.sun.java.jdk.win32.x86_1.6.0.013/lib/tools.jar;/D:/Users/sll/AppDa
xwork.NullHandler.createNullObjects	false
current.property.path	null
attr	org.apache.struts2.util.AttributeMap@39d072
com.opensymphony.xwork2.ActionContext.container	com.opensymphony.xwork2.inject.ContainerImpl@1b6956f
com.opensymphony.xwork2.ActionContext.session	{user_name=aaa}
com.opensymphony.xwork2.ActionContext.actionInvocation	com.opensymphony.xwork2.DefaultActionInvocation@1a422d9
com.opensymphony.xwork2.util.ValueStack.ValueStack	com.opensymphony.xwork2.ognl.OgnlValueStack@10a4a32
session	{user_name=aaa}
report.conversion.errors	false
xwork.MethodAccessor.denyMethodExecution	false
struts.actionMapping	org.apache.struts2.dispatcher.mapper.ActionMapping@1ee9cc3
com.opensymphony.xwork2.ActionContext.parameters	{password=[Ljava.lang.String;@3d4817, username=[Ljava.lang.String;@6d0085}

图 17.3　查看 ValueStack 和 StackContext 中的所有信息

3. 任务总结

本任务使用 property 标签输出 ValueStack 和 StackContext 中的值，使用 property 标签输出字符串、HTML 标签和设置默认值，使用 debug 标签查看 ValueStack 和 StackContext 中的值。

17.2.4　bean 标签和 param 标签

bean 标签用于创建一个 JavaBean 的实例。param 标签为其他标签添加参数。因为 param 标签经常作为 bean 标签的子标签，为实例的属性赋值，因此将两个标签放在一起讲解。

bean 标签常用属性有两个，各属性及说明如表 17.2 所示。

表 17.2　bean 标签的属性及说明

属性名	是否必需	默认值	类 型	说　　明
name	true		String	用来指定该实例对应的类
var	false		String	用于设置实例在值栈中的名字，并将实例放入 StackContext 中，当不设置此属性时实例会被放入 ValueStack 的栈顶，标签结束后移出 ValueStack

var 属性也可以使用 id 属性来代替，但推荐使用 var 属性，id 属性以后将被弃用。

另外，需要注意的是：因为 bean 标签不设置 var 属性时，标签结束后 bean 标签所创建的实例就被移出 ValueStack，该实例就无法访问了，因此不设置 var 属性时，只能在标签结束前访问实例，或者设置 var 属性，通过 StackContext 访问该实例。

param 标签常用属性有两个，各属性及说明如表 17.3 所示。

表 17.3　param 标签的属性及说明

属性名	是否必需	默认值	类 型	说　　明
name	false		String	用来指定 param 标签定义的属性的名称
value	false		String	用来指定属性的值

param 标签有两种使用方式，格式如下。

(1) 第一种方式：

```
<s:param name="属性名" value="属性值">
```

(2) 第二种方式：

```
<s:param name="属性名">属性值</s:param>
```

需要注意的是：当使用第一种方式时，属性值被看做 OGNL 表达式，因此如果属性值为字符串，需要加单引号。

在 Struts 2 中定义一个类 User，代码如下：

```
package com.cvit.struts2.user.action;

public class User {
    private String name;
    private String password;
    //属性的 getter 和 setter 方法略
    ...
}
```

在 JSP 文件中，使用 bean 标签将 User 类实例化的代码如下：

```
<s:bean name="com.cvit.struts2.user.action.User">
    <s:param name="name" value="'Tom'"></s:param>
    <s:param name="password">123</s:param>
    <br/>name=<s:property value="name"/>
```

```
    <br/>password=<s:property value="password"/>
</s:bean>
```

bean 标签的 name 属性指定被实例化的类是 com.cvit.struts2.user.action 包下的 User 类。

param 为该实例的 name 属性赋值为字符串"Tom"，password 属性赋值为字符串"123"。

因为 bean 标签未设置 var 属性，所以访问实例的属性值的 property 标签必须写在 bean 标签内。页面显示 User 类实例的 name 值和 password 值，如图 17.4 所示。

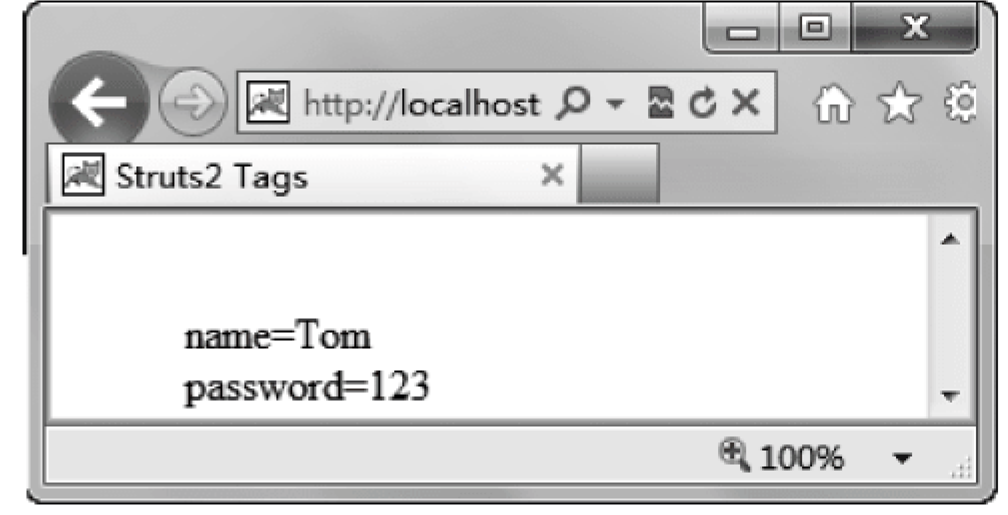


图 17.4 使用 bean 标签

使用 bean 标签还可以设置 var 属性，将创建的实例放入 StackContext 中，就可以在 bean 标签外访问实例了，代码如下：

```
<s:bean name="com.cvit.struts2.user.action.User" var="user">
    <s:param name="name" value="'Tom'"></s:param>
    <s:param name="password">123</s:param>
</s:bean>
#user.name=<s:property value="#user.name"/>
```

页面显示#user.name=Tom。

17.2.5 实践任务 2：使用 bean 标签和 param 标签在页面将类实例化

1. 任务说明

本任务实践使用 bean 标签将指定类实例化，并使用 param 标签为实例的指定属性赋值。

2. 任务实施

(1) 打开工程。打开工程 BBS_Struts2_1500_tags。

(2) 创建 User 类。创建 User 类，代码如下：

```
package com.cvit.struts2.user.action;

public class User {
    private String name;
    private String password;
    //属性的 getter 和 setter 方法略
    ...
}
```

(3) Action 类。LoginAction 代码不需要修改。

(4) 配置 Action。配置文件 struts.xml 代码不必修改。

(5) 编写视图层，插入 bean 标签和 param 标签。修改 index.jsp，主要代码如下：

```
<body>
    <ol>
        ...
        <li>
            定义 bean,并使用 param 来设定新的属性值:
            <s:bean name="com.cvit.struts2.user.action.User">
                <s:param name="name" value="'Tom'"></s:param>
                <s:param name="password">123</s:param>
                <br/>name=<s:property value="name"/>
                <br/>password=<s:property value="password"/>
            </s:bean>
            <br/>在 bean 标签外输出 password=<s:property value="password"/>
        </li>

        <li>
            定义 bean,并使用 var 来将实例放入 StackConntext 中:
            <s:bean name="com.cvit.struts2.user.action.User" var="user">
                <s:param name="name" value="'Tom'"></s:param>
                <s:param name="password">123</s:param>
            </s:bean>
            #user.name=<s:property value="#user.name"/>
        </li>
    </ol>
</body>
```

(6) 测试。部署工程，在浏览器的地址栏中输入 http://localhost:8080/BBS_Struts2_1500_tags/login，打开 index.jsp 页面，在第 6 行显示实例的 name 属性值和 password 属性值，但是在 bean 标签外显示不出 password 的属性值，在第 7 行显示 user 对象的 name 属性值，如图 17.5 所示。



图 17.5　index.jsp 页面 2

3. 任务总结

本任务使用 bean 标签实例化类，并使用 param 标签对实例的指定属性进行赋值，当未指定 bean 标签的 var 属性时，只能在 bean 标签内访问实例的属性，当指定 bean 标签的 var 属性时，可以使用"#对象名.属性名"的方法访问实例的属性。

17.2.6 set 标签

set 标签用来创建变量，并把变量放入指定范围，例如 application、session 等。

当某个值所在对象图深度非常深时，例如：user.post.theme.name，每次访问该值不仅性能低下，而且代码可读取也差。为避免这个问题，可以将该值赋给一个变量，并将变量放到特定范围内。

set 标签常用属性有三个，各属性及说明如表 17.4 所示。

表 17.4 set 标签的属性及说明

属性名	是否必需	默认值	类 型	说 明
scope	false	action	String	指定变量被放置的范围，可以是 application、session、request、page 或 action。该属性的默认值是 action 和 request
value	false		String	指定变量值
var	false		String	指定变量名，如果设置了该属性，会将变量放入 Value Stack 中

var 也可以使用 id 或 name 代替，但推荐使用 var，id 和 name 以后会被弃用。

下面我们通过一个实例来介绍 set 标签的用法，代码如下：

```
<li>set 设定 username.length()的值(默认为 request 和 ActionContext): <s:set var=
"nameLength" value=" username.length()" /></li>
<li>set 从 request 取值: <s:property value="#request. nameLength" /></li>
<li>set 从 ActionContext 取值: <s:property value="# nameLength" /></li>
<li>set 设定范围: <s:set name="passLength" value="password.length()" scope="page"/>
</li>
<li>set 从相应范围取值: <%=pageContext.getAttribute("passLength") %></li>
```

当不设置 scope 时，变量 nameLength 所在的范围是 action 和 request，action 即 Action Context，取变量 nameLength 的值可使用 OGNL 表达式#request. nameLength 或# nameLength。

当设置 scope="page"时，变量 passLength 被放入 page 中，取变量 passLength 的值的代码为<%=pageContext.getAttribute("passLength") %>。

17.2.7 实践任务 3：使用 set 标签设置变量

1. 任务说明

本任务实践使用 set 标签设置变量，并将变量放入指定范围。

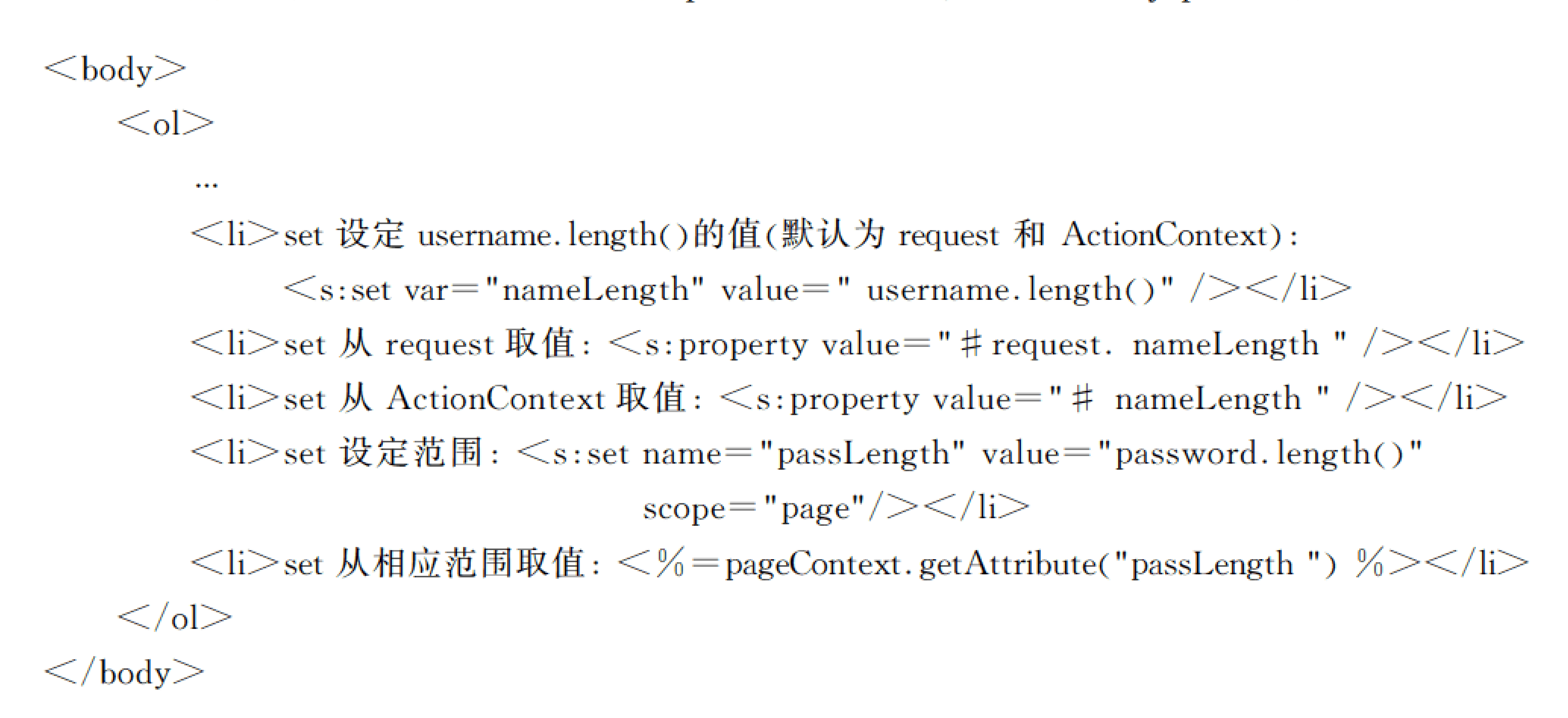
2. 任务实施

(1) 打开工程。打开工程 BBS_Struts2_1500_tags。

(2) Action 类。LoginAction 代码不必修改。

(3) 配置 Action。配置文件 struts.xml 代码不需要修改。

(4) 编写视图层,插入 bean 标签和 param 标签。修改 index.jsp,主要代码如下:



```
<body>
    <ol>
        ...
        <li>set 设定 username.length()的值(默认为 request 和 ActionContext):
            <s:set var="nameLength" value=" username.length()" /></li>
        <li>set 从 request 取值: <s:property value="#request. nameLength " /></li>
        <li>set 从 ActionContext 取值: <s:property value="# nameLength " /></li>
        <li>set 设定范围: <s:set name="passLength" value="password.length()"
                        scope="page"/></li>
        <li>set 从相应范围取值: <%=pageContext.getAttribute("passLength ") %></li>
    </ol>
</body>
```

(5) 测试。部署工程,在浏览器的地址栏中输入 http://localhost:8080/BBS_Struts2_1500_tags/login.jsp,打开 login.jsp 页面,输入用户名为"abcde"、密码为"123",单击"登录 1"按钮,打开 index.jsp 页面,显示用户名和密码的长度,如图 17.6 所示。

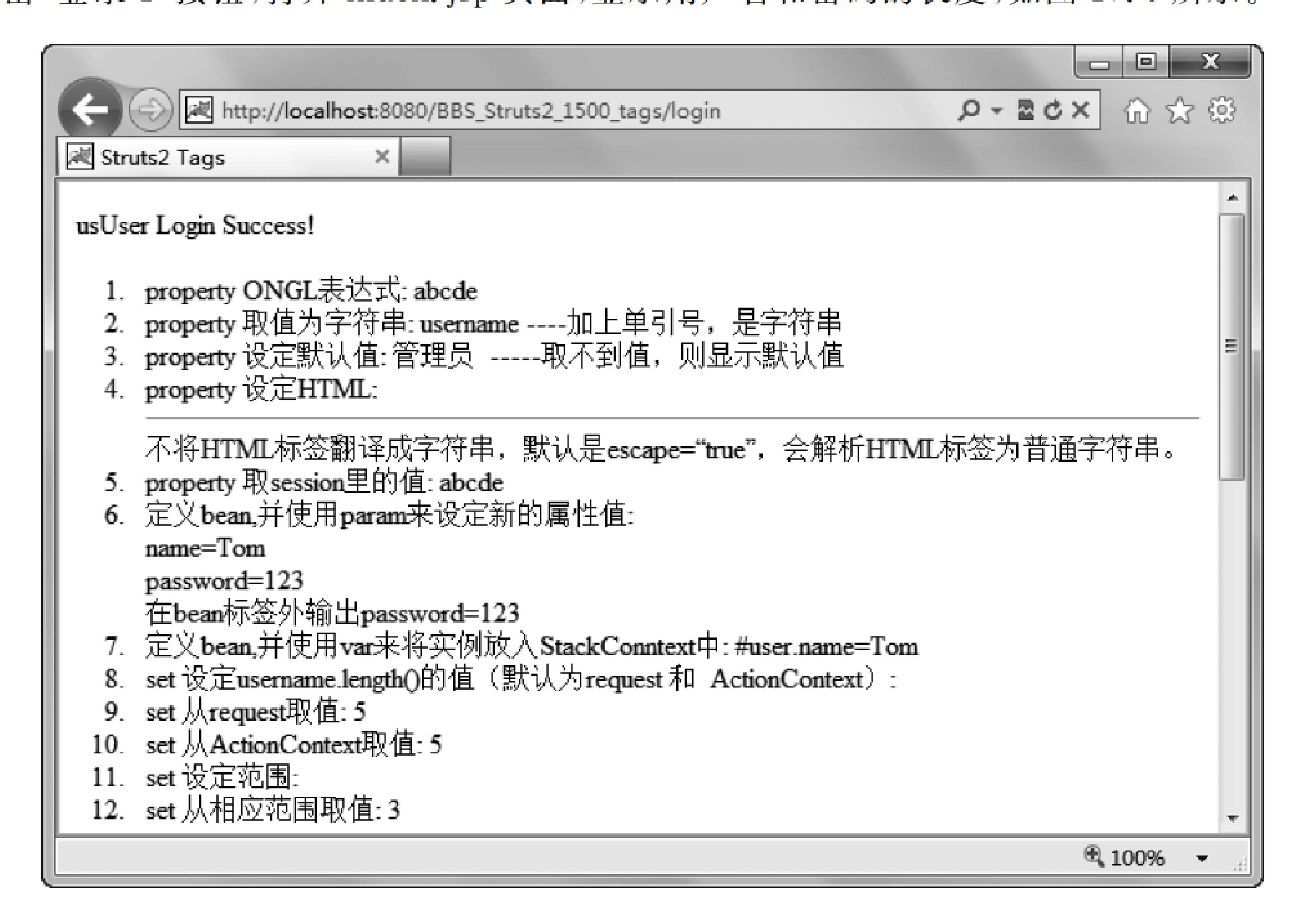


图 17.6 index.jsp 页面 3

3. 任务总结

本任务使用 set 标签设置变量，并放入指定范围。

17.2.8 date 标签

date 标签用于输出一个日期，并可以指定这个日期的输出格式。

date 标签常用属性有三个，各属性及说明如表 17.5 所示。

表 17.5 date 标签的属性及说明

属性名	是否必需	默认值	类　型	说　　明
format	false		String	用于指定日期的输出格式。格式类似于 yyyy/MM/dd/hh/mm/ss，其中 yyyy 代表年，MM 代表月，dd 代表日，hh 代表小时，mm 代表分，ss 代表秒。默认格式是 yyyy-MM-dd hh:mm:ss
name	true		String	指定变量值
var	false		String	指定变量名，如果设置了该属性，会将变量放入 Value Stack 中

date 标签的使用在第 12 章的数据类型转换中已经讲解，此处不再赘述。

17.3 控制标签

控制标签主要用来控制输出的流程，实现分支、循环等控制。

17.3.1 if 标签、else if 标签、else 标签

if 标签、else if 标签和 else 标签都用于进行分支控制，根据 test 属性值来决定是否计算、输出标签体的内容。

if 标签和 else if 标签都有个 test 属性，该属性说明如表 17.6 所示。

表 17.6 if 标签和 else if 标签的 test 属性及说明

属性名	是否必需	默认值	类　型	说　　明
test	true		boolean	用于判断是否计算、输出该标签体的内容

这 3 个标签可以组合使用，只有 if 标签可以单独使用，其他两个标签必须与 if 标签结合使用，标签也可以互相嵌套使用。标签的用法与 JSP 脚本的 if 条件语句用法相同。test 属性值为 boolean 类型，当值为真时，计算、输出 if 标签体的内容，否则计算、输出 else 标签体的内容。

下面是示例代码。

```
<s:if test="username.length()<2">用户名太短!</s:if>
    <s:elseif test="username.length()>10">用户名太长!</s:elseif>
        <s:else>用户名正确!</s:else>
```

当 username 的长度小于 2 时，输出“用户名太短!”；当 username 的长度大于 10 时，输出“用户名太长!”；当 username 长度在 2 到 10 之间时，输出“用户名正确!”。

17.3.2　实践任务 4：使用 if 标签、else if 标签、else 标签实现分支控制

1. 任务说明

本任务实践使用 if 标签、else if 标签、else 标签实现分支控制。

2. 任务实施

（1）打开工程。打开工程 BBS_Struts2_1500_tags。

（2）Action 类。LoginAction 代码不需要修改。

（3）配置 Action。配置文件 struts.xml 代码不必修改。

（4）编写视图层，插入 if 标签、else if 标签、else 标签。修改 index.jsp，主要代码如下：

```
<body>
    <ol>
        …
        <li>
            <s:if test="username.length()<2">用户名太短!</s:if>
            <s:elseif test="username.length()>10">用户名太长!</s:elseif>
            <s:else>用户名正确!</s:else>
        </li>
    </ol>
</body>
```

（5）测试。部署工程，在浏览器的地址栏中输入 http://localhost:8080/BBS_Struts2_1500_tags/login.jsp，打开 login.jsp 页面。

① 输入用户名为“a”、密码为“123”，单击“登录 1”按钮，打开 index.jsp 页面，显示“用户名太短!”。

② 输入用户名为“abcde12345678”、密码为“123”，单击“登录 1”按钮，打开 index.jsp 页面，显示“用户名太长!”。

③ 输入用户名为“abcde”、密码为“123”，单击“登录 1”按钮，打开 index.jsp 页面，显示“用户名正确!”。

3. 任务总结

本任务使用 if 标签、else if 标签、else 标签实现分支控制，根据 test 属性值判断是否计算、输出标签体内容。

17.3.3 iterator 标签

iterator 标签主要用于对集合进行迭代，这个集合可以是一个 List、Map 或数组，也可以是一个 Set 类型。该标签常用属性有三个，各属性及说明如表 17.7 所示。

表 17.7 iterator 标签属性及说明

属性名	是否必需	默认值	类 型	说 明
value	false		String	用于指定被迭代的迭代体，此属性值可以以集合形式给出，可以是值栈中的集合，也可以是一个 OGNL 表达式，其默认值是 ValueStack 的栈顶
var	false		String	用于指定集合中当前被迭代的元素在 Stack Context 中的名称，当本次循环结束后当前迭代元素将被移出 Stack Context
status	false		String	指定迭代时的 IteratorStatus 实例。通过该实例就可以判断当前迭代元素的属性

下面是迭代输出示例代码。

```
<s:iterator value="{1,2,3}">
     <s:property />|
</s:iterator>
```

上例是使用 OGNL 表达式获得数组{1,2,3}，<s:property />迭代输出数组中的每个元素，在页面输出“1|2|3|”。

迭代输出 Map 类型集合的示例代码如下：

```
<s:iterator value="#{1:'a', 2:'b', 3:'c'}">
     <s:property value="key" /> | <s:property value="value" />
     <br />
</s:iterator>
```

输出结果如下：

```
1 | a
2 | b
3 | c
```

iterator 标签还可以定义 var 属性，将当前被迭代的元素放入 Stack Context，并使用 OGNL 表达式访问元素，示例代码如下：

```
<s:iterator value="{'aaa','bbb','ccc'}" var="x">
     <s:property value="#x.toUpperCase()" />|
</s:iterator>
```

x 代表当前被迭代的元素，使用 toUpperCase()方法将字符串中的字母转换为大写字母，输出结果如下：

AAA| BBB| CCC|

例如在论坛管理系统中，从数据库读取所有用户信息，保存在 List 集合 userList 中。若用户类有 name 属性，要迭代输出所有用户的 name 属性值，代码如下：

```
<s:iterator value="userList" var="user">
    <s:property value="#user.name " /><br />
</s:iterator>
```

如果为 iterator 标签指定 status 属性，即每次迭代时会有一个 IteratorStatus 实例，该实例包含如下几个方法。

(1) int getCount()：返回当前迭代了几个元素。

(2) int getIndex()：返回当前迭代元素的索引。

(3) boolean isEven()：返回当前被迭代元素的索引是否是偶数。

(4) boolean isOdd()：返回当前被迭代元素的索引是否是奇数。

(5) boolean isFirst()：返回当前被迭代元素是否是第一个元素。

(6) boolean isLast()：返回当前被迭代元素是否是最后一个元素。

17.3.4 实践任务 5：使用 iterator 标签迭代集合

1. 任务说明

本任务实践使用 iterator 标签实现对数组、Map 集合的迭代，并使用 status 属性获得当前被迭代元素的索引属性。

2. 任务实施

(1) 打开工程。打开工程 BBS_Struts2_1500_tags。

(2) Action 类。LoginAction 代码不需要修改。

(3) 配置 Action。配置文件 struts.xml 代码不必修改。

(4) 编写视图层，插入 iterator 标签。修改 index.jsp，主要代码如下：

```
<body>
    <ol>
        ...
        <li>
            遍历数组类型的集合：
            <s:iterator value="{1,2,3}">
                <s:property /> |
            </s:iterator>
        </li>

        <li>
            遍历数组类型的集合，定义 var：
            <s:iterator value="{'aaa', 'bbb', 'ccc'}" var="x">
                <s:property value="#x.toUpperCase()" />|
            </s:iterator>
```

```
        </li>

        <li>
            遍历 Map 类型集合:
            <s:iterator value="#{1:'a', 2:'b', 3:'c'}">
                <s:property value="key" /> | <s:property value="value" />
                <br />
            </s:iterator>
        </li>

        <li>
            遍历 Map 类型集合,定义 var:
            <s:iterator value="#{1:'a', 2:'b', 3:'c'}" var="x">
                <s:property value="#x.key" /> | <s:property value="#x.value" />
                <br>
            </s:iterator>
        </li>
        <li>
            使用 status:
            <br>
            <s:iterator value="{'aaa', 'bbb', 'ccc'}" status="status">
                <s:property /> |
                历过的元素总数: <s:property value="#status.count" /> |
                遍历过的元素索引: <s:property value="#status.index" /> |
                当前是偶数?: <s:property value="#status.even" /> |
                当前是奇数?: <s:property value="#status.odd" /> |
                是第一个元素吗?: <s:property value="#status.first" /> |
                是最后一个元素吗?: <s:property value="#status.last" />
                <br>\
            </s:iterator>
        </li>
    </ol>
</body>
```

(5) 测试。部署工程,在浏览器的地址栏中输入 http://localhost:8080/BBS_Struts2_1500_tags/login,打开 index.jsp 页面,迭代集合,如图 17.7 所示。

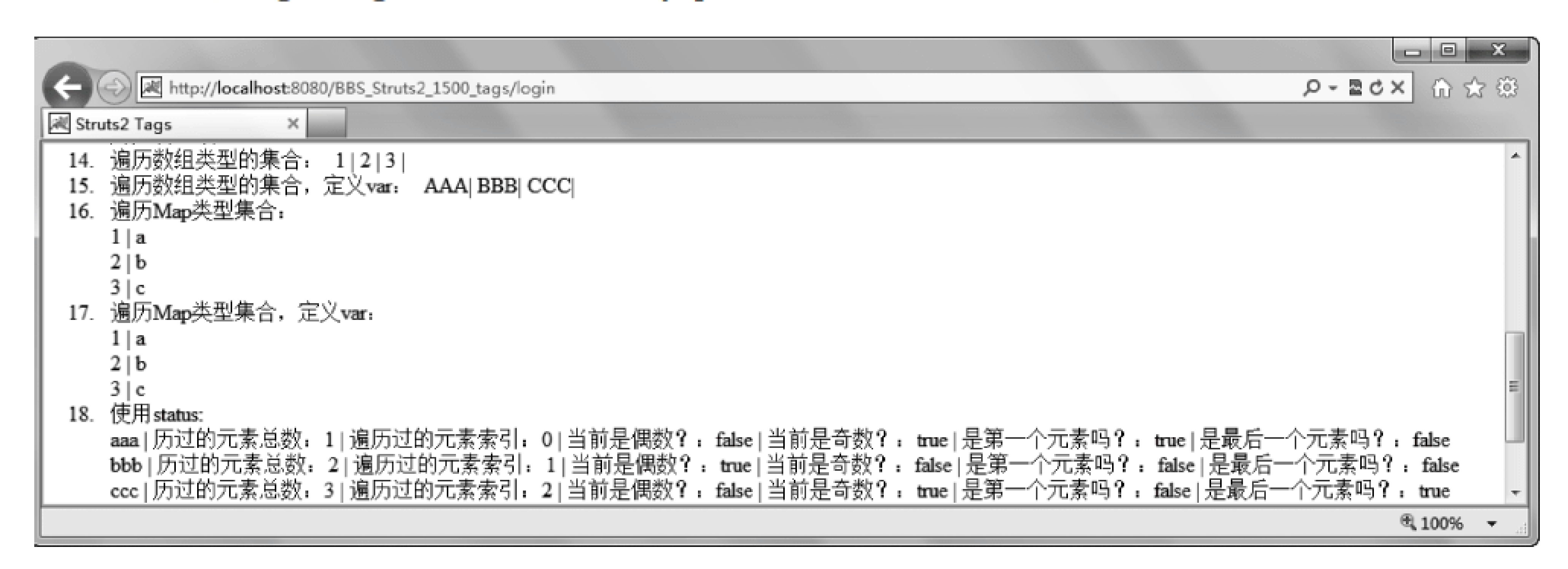


图 17.7　index.jsp 页面 4

3. 任务总结

本任务使用 iterator 标签实现循环控制,迭代输出集合,并输出被迭代元素的索引信息。

17.3.5　实践任务 6: 嵌套使用 if 标签、else 标签和 iterator 标签

1. 任务说明

在论坛管理系统中,从数据库中读取所有用户信息,如果无用户,在页面显示"无记录",如果有用户,在页面列表显示用户名。

2. 任务实施

(1) 打开工程。打开工程 BBS_Struts2_1500_tags。

(2) Action 类。在 Action 类中定义 User 类型的 List 集合 users,代码如下:

```
package com.cvit.struts2.user.action;

import java.util.ArrayList;
import java.util.List;
import org.apache.struts2.interceptor.SessionAware;
import com.opensymphony.xwork2.ActionSupport;

public class LoginAction extends ActionSupport implements SessionAware{
    private String username;
    private String password;
    private Map<String, Object> session;
    private List<User> users;
    public String execute() {
        users = new ArrayList<User>();
        session.put("user_name", username);
        return SUCCESS;
    }
    //属性的 getter 和 setter 方法略
}
```

在实际开发中,LoginAction 的 execute()方法将调用业务逻辑组件,从数据库中读取所有用户的信息,并保存到 users 中。本任务主要实践 Struts 标签,因此此处不调用业务逻辑组件,用 new ArrayList<User>()语句创建空的 users 集合,代表在数据库中未查到任何记录。

(3) 配置 Action。配置文件 struts.xml 代码不需要修改。

(4) 编写视图层,插入 iterator 标签。修改 index.jsp,主要代码如下:

```
<body>
    <ol>
        ...
        <li>
                <s:if test="users.size()<1">无记录</s:if>
                <s:else>
```

```
                <s:iterator value="users" var="user">
                    <s:property value="#user.name" /> |
                    <s:property value="#user.password" />
                    <br />
                </s:iterator>
            </s:else>
        </li>
    </ol>
</body>
```

(5) 测试。部署工程，在浏览器的地址栏中输入 http://localhost:8080/BBS_Struts2_1500_tags/login，打开 index.jsp 页面，迭代集合 users，如图 17.8 所示。

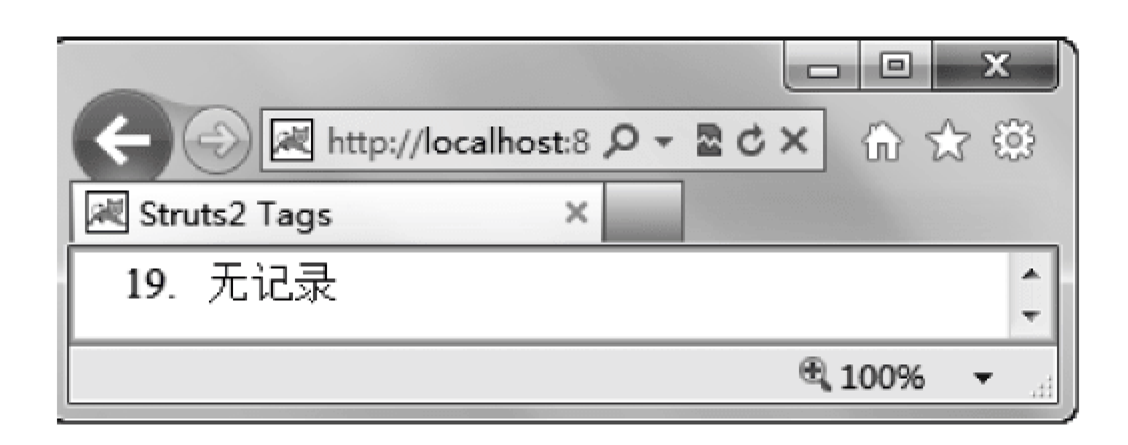


图 17.8　index.jsp 显示未查找到记录

(6) Action 类。在 Action 类中向 users 集合中插入元素，代码如下：

```
package com.cvit.struts2.user.action;

import java.util.ArrayList;
import java.util.List;
import org.apache.struts2.interceptor.SessionAware;
import com.opensymphony.xwork2.ActionSupport;

public class LoginAction extends ActionSupport implements SessionAware{
    private String username;
    private String password;
    private Map<String, Object> session;
    private List<User> users;
    public String execute() {
        users = new ArrayList<User>();
        users.add(new User("n1","p1"));
        users.add(new User("n2","p2"));
        users.add(new User("n3","p3"));
        session.put("user_name", username);
        return SUCCESS;
    }
    //属性的 getter 和 setter 方法略
}
```

在实际开发中，LoginAction 的 execute()方法将调用业务逻辑组件，从数据库中读取所有用户的信息，并保存到 users 中。本任务主要实践 Struts 标签，因此此处不调用业务逻辑组件，用 add()方法向集合中添加元素，表示在数据库中查到三条记录。

(7) 测试。部署工程，在浏览器的地址栏中输入 http://localhost:8080/BBS_Struts2_1500_tags/login，打开 index.jsp 页面，迭代集合 users，如图 17.9 所示。

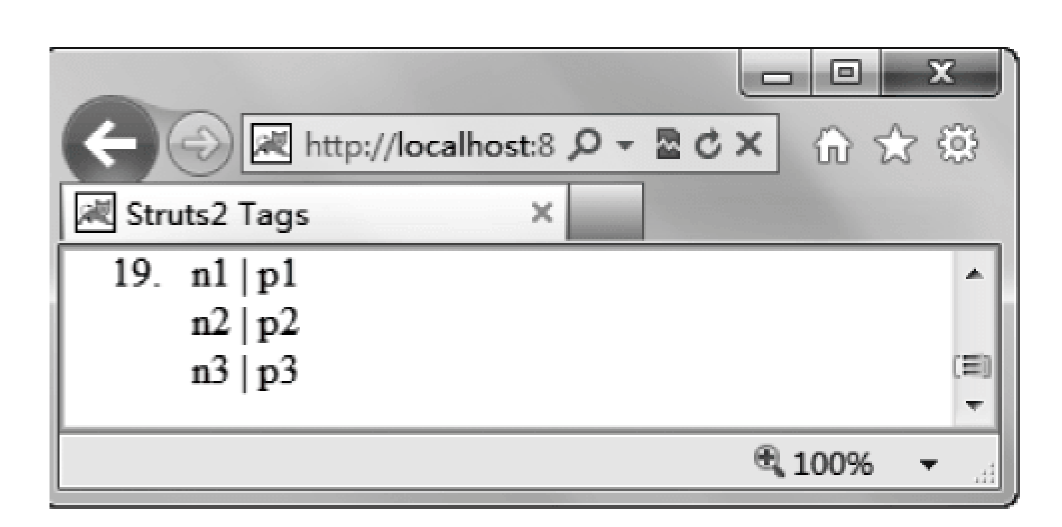


图 17.9　index.jsp 列表显示用户信息

3. 任务总结

本任务使用 if 标签、else 标签和 iterator 标签模拟实现列表显示用户信息。

17.3.6　拓展任务 1：实现后台用户管理模块中用户信息列表显示

1. 任务要求

实现从列表显示所有用户的用户名、用户性别、年龄等信息。

2. 建议步骤

(1) 创建工程。

(2) 创建视图层文件 User_list_success.jsp。

(3) 创建 User 类，定义属性。

(4) 创建 UserAction 类，定义属性为由 User 类型的元素组成的 List 集合，创建 list() 方法。

(5) 编写 struts.xml 文件，配置 Action。

(6) 打开 User_list_success.jsp 文件编写列表显示所有用户信息的代码。

(7) 部署工程。

(8) 调试系统。打开浏览器，输入客户端请求，测试程序，如有错误，进行修改。

17.3.7　拓展任务 2：实现后台主题管理模块中主题信息列表显示

1. 任务要求

实现从列表显示所有主题的主题名、主题说明、版主名等信息。

2. 建议步骤

(1) 创建工程。

(2) 创建视图层文件 Theme_list_success.jsp。

(3) 创建 User 类和 Theme 类，定义属性。

(4) 创建 ThemeAction 类，定义属性为由 Theme 类型的元素组成的 List 集合，创建 list()方法。

(5) 编写 struts.xml 文件，配置 Action。

(6) 打开 Theme_list_success.jsp 文件，编写列表显示所有主题信息的代码。

(7) 部署工程。

(8) 调试系统。打开浏览器，输入客户端请求，测试程序，如有错误，进行修改。

17.4 表单标签

Struts 2 中常用的表单标签与 HTML 标签功能相同，用法也一样，而且 Struts 2 的表单标签实现数据传递更加方便，两者的主要区别就是标签的形式不同。

Struts 2 表单标签与 HTML 标签的对应关系如表 17.8 所示。

表 17.8 与 HTML 标签功能相同的 Struts 2 表单标签

HTML 标签	Struts 2 标签
<form>	<s:form>
<input type="textfield">	<s:textfield>
<input type="password">	<s:password>
<input type="radio">	<s:radio>
<input type="checkbox">	<s:checkbox>
<input type="file">	<s:file>
<input type="hidden">	<s:hidden>
<input type="submit">	<s:submit>
<input type="reset">	<s:reset>
<input type="textarea">	<s: textarea >
<input type="select">	<s:select>

下面通过一个综合实例来演示标签的使用，页面主要代码如下：

```
<body>
    <s:form action="">
        <s:textfield name="user.name" label="textfield 标签" />
        <s:password name="user.password" label="password 标签" />
        <s:radio name="sex" list="#{'man':'男','woman':'女'}" label="radio 标签" />
        <s:checkbox value="checkbox" name="user.interest" label="checkbox 标签" />
        <s:file name="user.file" label="file 标签" />
        <s:hidden name="user.id" label="hidden 标签" />
        <s:textarea name="user.address" label="textarea 标签" />
        <s:select list="#{'1':'选项一','2':'选项二','3':'选项三'}" label="select 标签" />
        <s:submit value="确定" />
        <s:reset value="重写" />
    </s:form>
</body>
```

当此页面被请求时，浏览器的显示如图 17.10 所示。

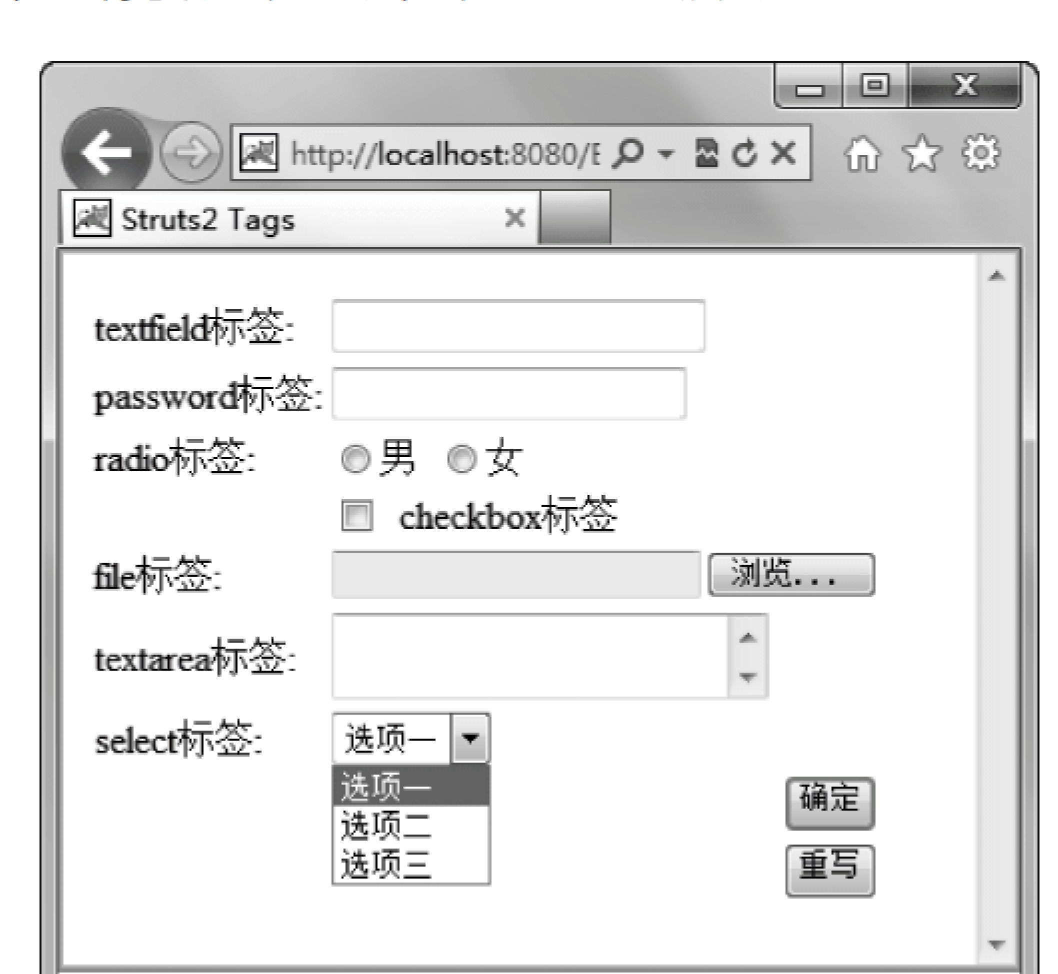


图 17.10　使用与 HTML 相同的标签

17.5　非表单 UI 标签

非表单 UI 标签主要用于生成非表单性质的可视化元素。例如在页面显示 Action 中封装信息的标签，主要包括 fielderror 标签、actionerror 标签、actionmessage 标签等。

17.5.1　fielderror 标签

fielderror 标签用于输出 fieldErrors 中的值。在进行类型转换和输入校验发生错误时，经常使用此标签。

<s:fielderror />输出 fieldErrors 中所有的值。

如果需要输出指定字段的错误信息，需要在 fielderror 标签中指定字段名称，其实现方式主要有以下两种。

```
<s:fielderror fieldName="字段名" />
```

或

```
<s:fielderror>
    <s:param>字段名</s:param>
</s:fielderror>
```

下面我们通过一个实例来学习 fielderror 标签的使用方法。

首先编写 Action 类，用来向 fieldErrors 中添加信息，该类的主要代码如下：

```
public class FieldErrorAction extends ActionSupport{
    public String execute () {
    //向 username 字段添加错误信息
```

```
            this.addFieldError("username", "用户名不能为空");
            //向 password 字段添加错误信息
            this.addFieldError("password", "密码不能为空");
            return ERROR;
        }
    }
```

在 struts.xml 中添加配置信息，代码如下：

```
<action name="fielderror" class="com.cvit.struts2.user.action.FieldErrorAction">
    <result>/fielderror.jsp</result>
</action>
```

在页面使用 fielderror 标签输出错误信息，输出全部字段错误信息和输出指定字段的错误信息，代码如下：

```
<h1>输出 fieldErrors 中所有错误信息：</h1>
<s:fielderror/>
<h1>输出 fieldErrors 中指定字段 username 的错误信息：</h1>
<s:fielderror fieldName="username"/>
<h1>输出 fieldErrors 中指定字段 password 的错误信息：</h1>
<s:fielderror>
    <s:param>password</s:param>
</s:fielderror>
```

在浏览器中提交请求后，页面显示如图 17.11 所示。

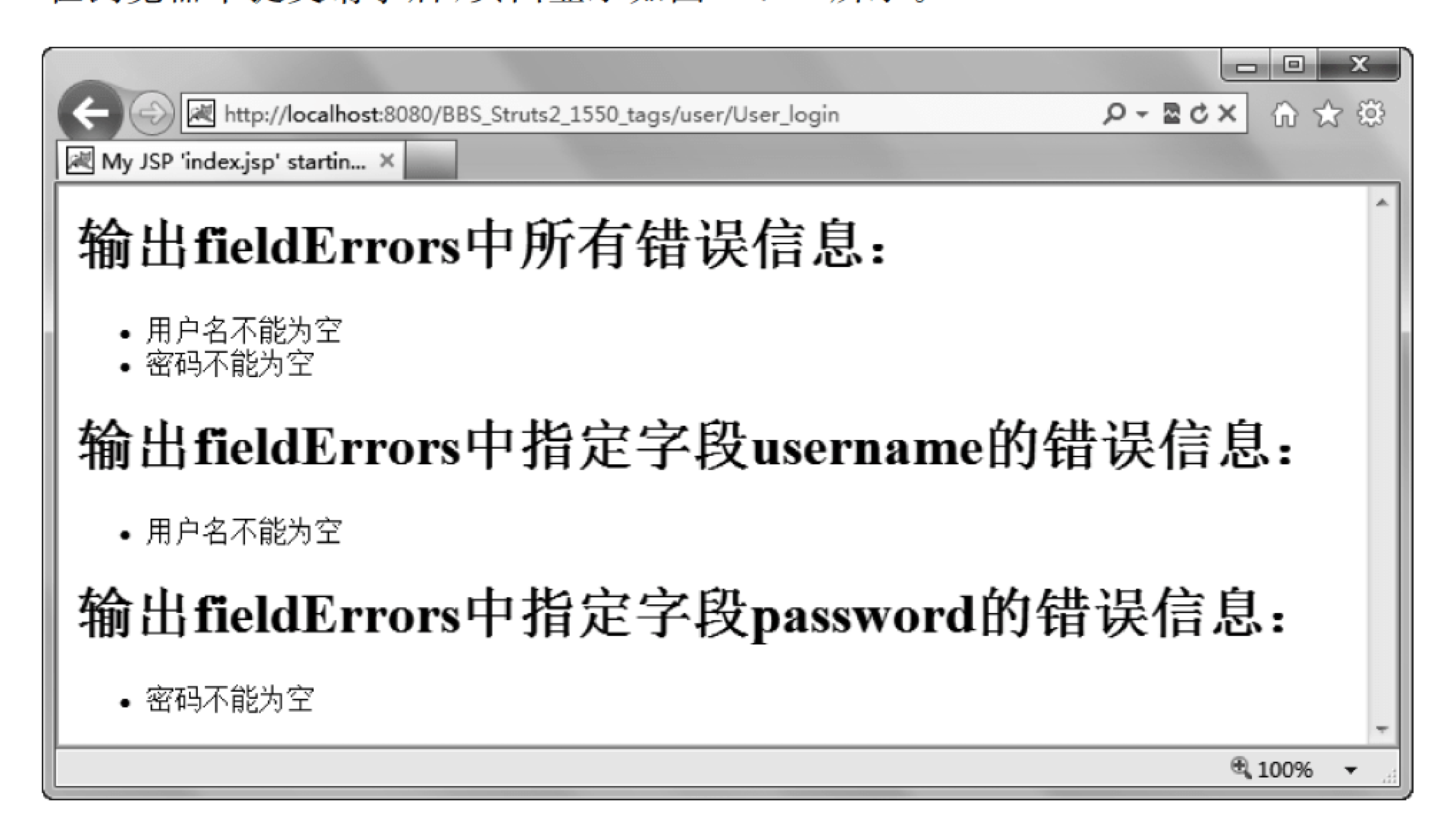


图 17.11 使用 fielderror 输出错误信息

17.5.2 actionerror 标签和 actionmessage 标签

actionerror 标签和 actionmessage 标签这两个标签用法相同，作用也几乎完全一样，都是负责输出 Action 实例中封装的信息，区别主要在于 actionerror 标签负责输出

actionErrors 中的信息，actionmessage 标签负责输出 actionMessages 中的信息。

下面我们通过一个实例来学习如何使用 actionerror 标签和 actionmessage 标签。

首先编写 Action 类，用来向 actionErrors 和 actionmessage 中添加信息，该类的主要代码如下：

```
public class ErrorMessageAction extends ActionSupport{
    public String execute () {
        //向 actionErrors 添加错误信息
        this.addActionError("这是 Action 中第一条错误信息");
        this.addActionError("这是 Action 中第二条错误信息");
        //向 actionmessage 字段添加错误信息
        this.addActionMessage("这是 Action 中第一条普通信息");
        this.addActionMessage("这是 Action 中第一条普通信息");
        return SUCCESS;
    }
}
```

在 struts.xml 中添加配置信息，代码如下：

```
<action name="actionErrorMessage" class="com.cvit.struts2.action.ErrorMessageAction">
    <result>/ actionErrorMessage.jsp</result>
</action>
```

在页面使用 actionerror 标签和 actionmessage 标签输出 Action 中的信息，代码如下：

```
<h1>输出 actionErrors 中的错误信息：</h1>
<s:actionerror/>
<h1>输出 actionMessages 中的普通信息：</h1>
<s:actionmessage/>
```

在浏览器中提交请求后，页面显示如图 17.12 所示。



图 17.12　actionerror 标签和 actionmessage 标签输出 Action 中的信息

当调用程序时，系统也会自动根据实际情况向 actionErrors 和 actionmessage 中添加信息，或者开发人员为查看程序执行情况，手动向 actionErrors 和 actionmessage 中添加

信息。使用 actionerror 标签和 actionmessage 标签可以查看这些信息，以帮助开发人员调试程序。

17.6 实训：实现论坛管理系统的用户管理模块

(1) 实现用户的登录、增、删、改、查等操作。
(2) 根据实际需求，在页面使用 struts 标签和 OGNL 表达式。
(3) 测试。

17.7 本章小结

本章从使用标签的原因讲起，介绍了 Struts 2 常用标签的分类，详细地讲解了以下标签的用途和具体使用方法。

(1) 数据标签：property 标签、debug 标签、bean 标签、param 标签、set 标签、date 标签。

(2) 控制标签：if 标签、else 标签、else if 标签、iterator 标签。

(3) 表单标签：form 标签、textfield 标签、password 标签、checkbox 标签、radio 标签、select 标签、textarea 标签、submit 标签、reset 标签。

(4) 非表单 UI 标签：fielderror 标签、actionerror 标签、actionmessage 标签。

本章重点讲解了 property 标签、debug 标签、if 标签、else 标签、else if 标签、iterator 标签的使用。

第18章

Struts 2 的拦截器

拦截器在 Struts 2 框架中具有重要作用，是整个 Struts 2 框架的核心和基础。Struts 2 的大多数核心功能的实现，如类型转换、输入校验、参数传递等，都要用到拦截器。Struts 2 框架可以看作一个空的容器，由大量的内建拦截器完成该框架的大部分操作。例如：params 拦截器将 HTTP 请求中的参数解析出来，设置成 Action 的属性；servletConfig 拦截器直接将请求中的 HttpServletRequest 实例和 HttpServletResponse 实例传给 Action，Action 访问 Servlet API 就是通过这个拦截器实现的；conversionError 拦截器将类型转换出现错误时的错误信息放到 Action 的错误字段集中；validation 拦截器通过执行在 xxxAction-validation.xml 中定义的校验器，从而完成数据校验……这些操作都是通过 Struts 2 内建拦截器完成的。

对于 Struts 2 框架的拦截器体系而言：当我们需要使用哪个拦截器时，只需在配置文件中应用该拦截器即可；如果不需要使用某拦截器，也只需在配置文件中取消该拦截器的应用——不管是否应用某个拦截器，对整个 Struts 2 框架不会有任何影响。这是一种可插拔式的设计，具有非常好的可扩展性。

Struts 2 框架的拦截器是动态配置的，所以开发人员可以应用 Struts 2 框架的内建拦截器，也可以取消对某内建拦截器的应用，开发人员还可以自定义拦截器，应用到框架中，扩展 Struts 2 框架，这些实现都是非常简单而方便的，也提高了代码的重用性。

Struts 2 的拦截器是一种 AOP(面向切面编程)设计，它允许开发人员以一种简单的方式来进行 AOP 方式开发。

本章要点：

- 拦截器的原理和运行机制
- 常用内建拦截器的功能、原理及使用方法
- 拦截器栈的应用
- 开发自定义拦截器
- 配置自定义拦截器

18.1 Struts 2 拦截器简介

在开始学习使用 Struts 2 拦截器之前，需要先对 Struts 2 拦截器的原理和设计机制有所了解。

18.1.1 什么是拦截器

拦截器的官方定义是：开发者可以在拦截器中定义一些代码，当 Action 被请求时，在 Actin 执行之前或之后调用这些代码，也可以在 Action 执行前阻止其执行。

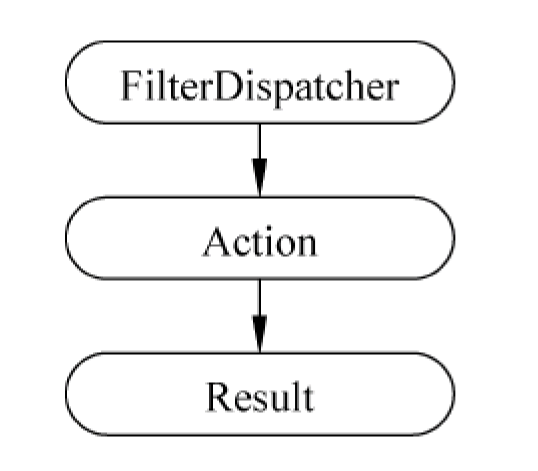


图 18.1 Action 执行流程

通俗一点来说，拦截器就是一个类，它以一种可插拔的方式，被定义在某个 Action 执行之前或之后执行，用来完成特定的功能。

例如，Action 需要进行权限验证和记录日志。在没使用拦截器时，Action 的执行流程如图 18.1 所示；使用拦截器后，Action 的执行流程如图 18.2 所示。

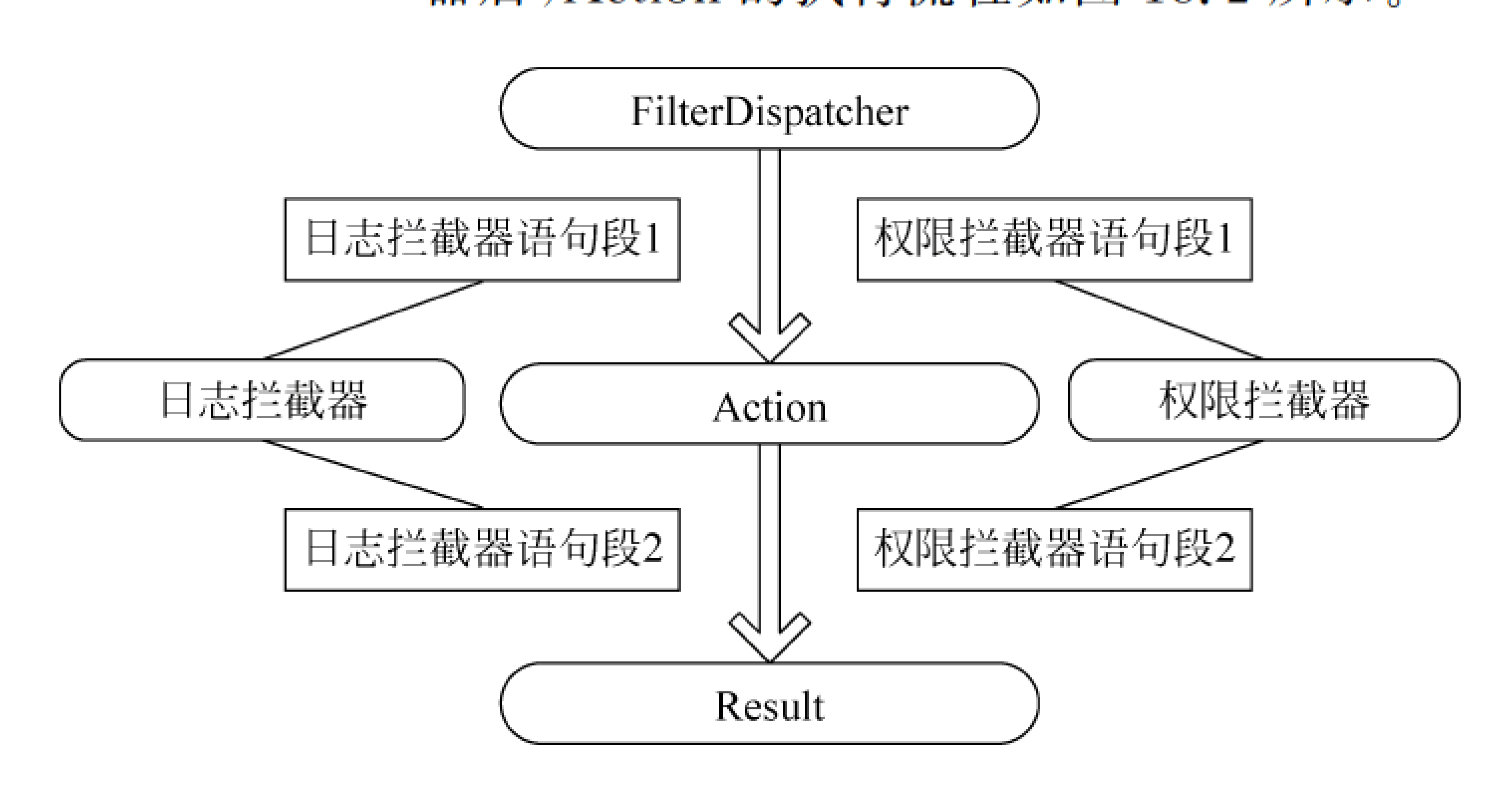


图 18.2 添加拦截器后 Action 的流程

图 18.2 所示的流程，实现了日志拦截器和权限验证拦截器，以可插拔的方式添加到 Action 的执行过程中，整个执行流程变为：执行日志拦截器语句段 1、执行权限验证拦截器语句段 1——执行 Action 中代码——执行权限验证拦截器语句段 2、执行日志拦截器语句段 2。

注意：拦截器代码段执行的顺序，在 Action 执行前后执行拦截器的顺序相反，如图 18.3 所示。

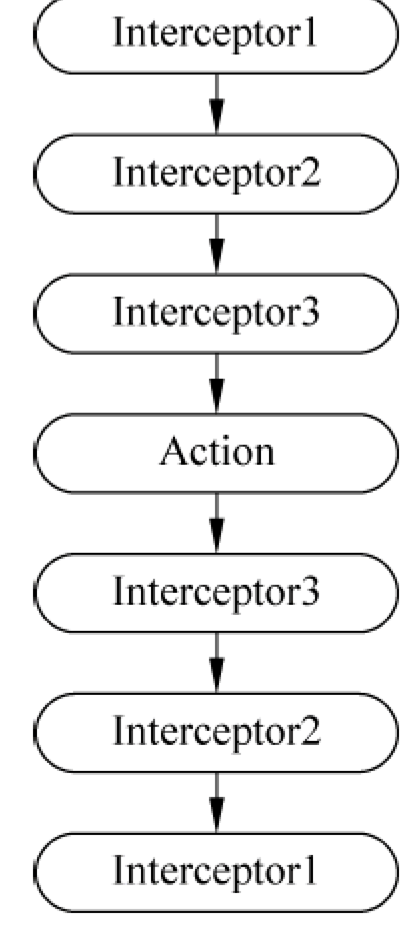


图 18.3 拦截器执行顺序

18.1.2 拦截器的设计机制

从根本上说，拦截器的设计是为了更好地实现代码重用。

例如上节的例子，权限验证和记录日志在一个 Web 应用中很多地方都会用到，如果不使用拦截器，可以通过在每个需要的 Action 中加入日志和权限验证的代码，或者将这两个方法写到类中，让 Action 继承类。但这两种方法都有缺点：使用第一种方法势必造成代码臃肿不堪，后期的代码

维护和修改也变得极为困难；使用第二种方法虽然在一定程度上减少了代码量，代码可维护性也得到了提高，但还是不够灵活。如果 Action 要增加或减少某个功能时，还需要修改很多代码，而采用拦截器方式可以解决以上问题。

我们知道拦截器是一个类，它将需要的功能封装在类中，当 Action 需要增加某功能时就配置拦截器，当 Action 需要减少某功能时就取消拦截器的配置。使用拦截器，当 Action 的功能发生变化时，只需修改配置就能轻松实现，使系统的开发、维护、修改的效率都得到了大大的提高。

18.2 Struts 2 内建拦截器的使用

Struts 2 有很多内建拦截器，这些拦截器可以在 struts-default.xml 配置文件中查看到。struts-default.xml 配置文件关于部署内建拦截器的代码段如下：

```
<interceptors>
  <interceptor name="alias" class="com.opensymphony.xwork2.interceptor.AliasInterceptor"/>
  <interceptor name="autowiring"
          class="com.opensymphony.xwork2.spring.interceptor.ActionAutowiringInterceptor"/>
  <interceptor name="chain"
          class="com.opensymphony.xwork2.interceptor.ChainingInterceptor"/>
  <interceptor name="conversionError"
          class="org.apache.struts2.interceptor.StrutsConversionErrorInterceptor"/>
  <interceptor name="cookie" class="org.apache.struts2.interceptor.CookieInterceptor"/>
  <interceptor name="clearSession"
          class="org.apache.struts2.interceptor.ClearSessionInterceptor" />
  <interceptor name="createSession"
          class="org.apache.struts2.interceptor.CreateSessionInterceptor" />
  <interceptor name="debugging"
          class="org.apache.struts2.interceptor.debugging.DebuggingInterceptor" />
  <interceptor name="externalRef"
          class="com.opensymphony.xwork2.interceptor.ExternalReferencesInterceptor"/>
  <interceptor name="execAndWait"
          class="org.apache.struts2.interceptor.ExecuteAndWaitInterceptor"/>
  <interceptor name="exception"
          class="com.opensymphony.xwork2.interceptor.ExceptionMappingInterceptor"/>
  <interceptor name="fileUpload"
          class="org.apache.struts2.interceptor.FileUploadInterceptor"/>
  <interceptor name="i18n"
          class="com.opensymphony.xwork2.interceptor.I18nInterceptor"/>
  <interceptor name="logger"
          class="com.opensymphony.xwork2.interceptor.LoggingInterceptor"/>
  <interceptor name="modelDriven"
          class="com.opensymphony.xwork2.interceptor.ModelDrivenInterceptor"/>
  <interceptor name="scopedModelDriven"
          class="com.opensymphony.xwork2.interceptor.ScopedModelDrivenInterceptor"/>
  <interceptor name="params"
          class="com.opensymphony.xwork2.interceptor.ParametersInterceptor"/>
```

```
<interceptor name="actionMappingParams"
            class="org.apache.struts2.interceptor.ActionMappingParametersInteceptor"/>
<interceptor name="prepare"
            class="com.opensymphony.xwork2.interceptor.PrepareInterceptor"/>
<interceptor name="staticParams"
            class="com.opensymphony.xwork2.interceptor.StaticParametersInterceptor"/>
<interceptor name="scope" class="org.apache.struts2.interceptor.ScopeInterceptor"/>
<interceptor name="servletConfig"
            class="org.apache.struts2.interceptor.ServletConfigInterceptor"/>
<interceptor name="sessionAutowiring"
            class="org.apache.struts2.spring.interceptor.SessionContextAutowiringInterceptor"/>
<interceptor name="timer"
            class="com.opensymphony.xwork2.interceptor.TimerInterceptor"/>
<interceptor name="token" class="org.apache.struts2.interceptor.TokenInterceptor"/>
<interceptor name="tokenSession"
            class="org.apache.struts2.interceptor.TokenSessionStoreInterceptor"/>
<interceptor name="validation"
            class="org.apache.struts2.interceptor.validation.AnnotationValidationInterceptor"/>
<interceptor name="workflow"
            class="com.opensymphony.xwork2.interceptor.DefaultWorkflowInterceptor"/>
<interceptor name="store" class="org.apache.struts2.interceptor.MessageStoreInterceptor" />
<interceptor name="checkbox" class="org.apache.struts2.interceptor.CheckboxInterceptor" />
<interceptor name="profiling"
            class="org.apache.struts2.interceptor.ProfilingActivationInterceptor" />
<interceptor name="roles" class="org.apache.struts2.interceptor.RolesInterceptor" />
<interceptor name="jsonValidation"
            class="org.apache.struts2.interceptor.validation.JSONValidationInterceptor" />
<interceptor name="annotationWorkflow"
            class="com.opensymphony.xwork2.interceptor.annotations.
                AnnotationWorkflowInterceptor" />
<interceptor name="multiselect"
            class="org.apache.struts2.interceptor.MultiselectInterceptor" />
...
</interceptors>
```

interceptor 的 name 属性指定拦截器名，class 属性指定拦截器的实现类。

struts-default.xml 配置文件中还配置了默认执行的拦截器，当任何 Action 被请求时，这些拦截器都默认被执行，struts-default.xml 文件中默认拦截器的配置代码片段如下：

```
<interceptor-stack name="defaultStack">
    <interceptor-ref name="exception"/>
    <interceptor-ref name="alias"/>
    <interceptor-ref name="servletConfig"/>
    <interceptor-ref name="i18n"/>
    <interceptor-ref name="prepare"/>
    <interceptor-ref name="chain"/>
    <interceptor-ref name="debugging"/>
    <interceptor-ref name="scopedModelDriven"/>
```

```
            <interceptor-ref name="modelDriven"/>
            <interceptor-ref name="fileUpload"/>
            <interceptor-ref name="checkbox"/>
            <interceptor-ref name="multiselect"/>
            <interceptor-ref name="staticParams"/>
            <interceptor-ref name="actionMappingParams"/>
            <interceptor-ref name="params">
                <param name="excludeParams">dojo\..*,^struts\..*</param>
            </interceptor-ref>
            <interceptor-ref name="conversionError"/>
            <interceptor-ref name="validation">
                <param name="excludeMethods">input,back,cancel,browse</param>
            </interceptor-ref>
            <interceptor-ref name="workflow">
                <param name="excludeMethods">input,back,cancel,browse</param>
            </interceptor-ref>
        </interceptor-stack>

        <!-- The completeStack is here for backwards compatibility for
             applications that still refer to the defaultStack by the
             old name -->
        <interceptor-stack name="completeStack">
            <interceptor-ref name="defaultStack"/>
        </interceptor-stack>
        ...
    <default-interceptor-ref name="defaultStack"/>
```

Struts 2 框架将默认执行的拦截器配置在拦截器栈 interceptor-stack 中，并使用<default-interceptor-ref>将拦截器栈中的拦截器设置为默认执行。

注意：关于拦截器栈和默认拦截器的内容，我们将在后面讲解。

下面我们以 exception 拦截器为例，讲解默认拦截器的使用方法。

18.2.1　Struts 2 默认内建拦截器的使用

在 Struts 2 框架默认拦截器中，名为 exception 的拦截器负责处理异常，当系统执行出现异常后，它将异常映射为结果。

例如，当系统访问数据库出现异常后，返回异常 java.lang.Exception 的实例，在配置文件中可以使用 exception 拦截器配置异常对应的逻辑视图，再使用 result 标签配置其物理视图，就可以在页面显示系统异常的信息。

下面示例使用 exception 拦截器。

首先创建 error_sql.jsp 文件，用于显示系统异常信息，主要代码如下：

```
<body>
    系统数据库访问出现异常,您可以联系我们,提交异常信息,我们将进行系统维护,尽快解决
    问题。谢谢支持!
</body>
```

然后在配置文件中配置异常映射结果，主要代码如下：

```
<action name="...">
<exception-mapping result="error" exception="java.lang.Exception"/>
<result name="error">/error_sql.jsp</result>
</action>
```

exception-mapping 标签的 exception 属性指定异常类型，result 属性配置逻辑视图，再使用<result>元素配置其物理视图为 error_sql.jsp，当系统访问数据库出现异常后，浏览器打开 error_sql.jsp，显示异常提示信息，如图 18.4 所示。

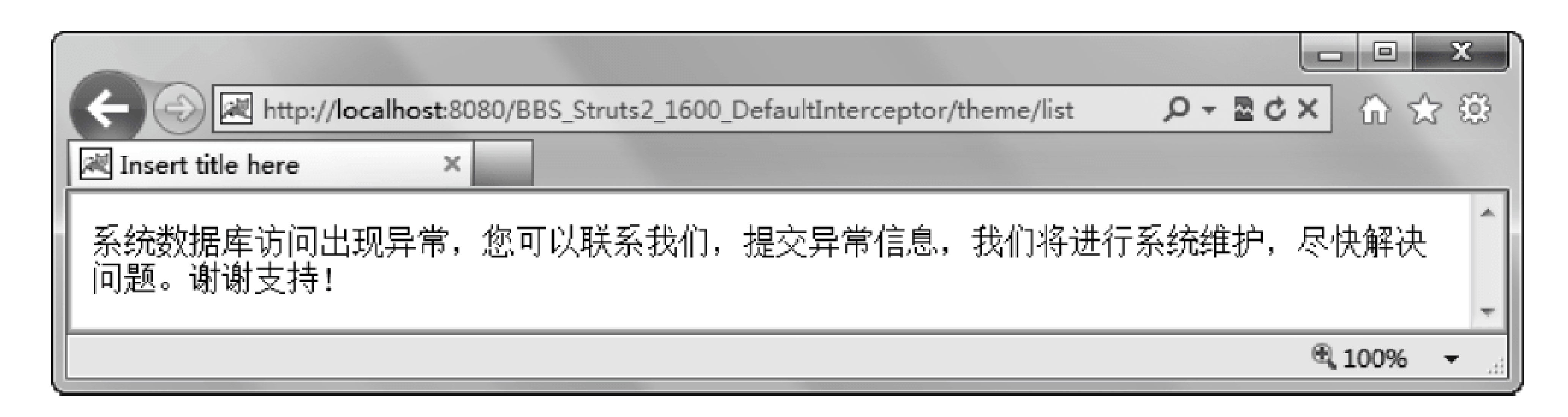


图 18.4 使用 exception 拦截器映射异常结果

大部分 Action 都需要处理数据库访问异常或其他异常，可以将映射配置为全局，写在公共包中，任何包中的 Action 需要实现映射，只需要继承公共包即可，示例代码如下：

```
<package name="BBS_default" extends="struts-default">
    <global-results>
        <result name="error">/error_sql.jsp</result>
    </global-results>
    <global-exception-mappings>
        <exception-mapping result="error" exception="java.sql.SQLException"/>
    </global-exception-mappings>
</package>
<package name="theme" extends="BBS_default" namespace="/theme">
        <action name="list" class="com.cvit.struts2.action.ThemeAction" method="list">
            <result>/Theme_list_success.jsp</result>
        </action>
        <action name="add" class="com.cvit.struts2.action.ThemeAction" method="add">
            <result>/Theme_add_success.jsp</result>
        </action>
...
</package>
```

将异常映射配置在名为 BBS_default 的<package>元素中，并定义为全局映射和全局结果集；在名为 theme 的<package>元素中，配置 extends="BBS_default"继承 BBS_default 包，则此包中所有 Action 访问数据库出现异常后，都跳转到 error_sql.jsp，显示数据库访问异常信息。

18.2.2 实践任务 1：使用 exception 拦截器实现声明式异常处理

1. 任务说明

本任务实践使用 exception 拦截器实现声明式异常处理。

在论坛管理系统中，从数据库中读取所有主题信息并显示主题信息列表，当访问数据库出现异常后，页面显示如图 18.4 所示的提示信息。

2. 任务实施

（1）创建工程。创建工程 BBS_Struts2_1600_DefaultInterceptor。

（2）视图层文件。

① 创建 index.jsp，主要代码如下：

```
<body> <a href="theme/list">主题列表</a></body>
```

② 创建 Theme_list_success.jsp，主要代码如下：

```
<body>
主题列表<br/>
<s:if test="themes.size==0">无记录</s:if>
    <s:else>
        <s:iterator value="themes" var="theme">
            主题名：<s:property value="#theme.name"/> |
            主题简述：<s:property value="#theme.descr"/><br/>
        </s:iterator>
    </s:else>
</body>
```

③ 创建 error_sql.jsp，主要代码如下：

```
<body>
    系统数据库访问出现异常，您可以联系我们，提交异常信息，我们将进行系统维护，尽快解决
    问题。谢谢支持！
</body>
```

（3）创建 Model 类。创建 Theme 类，代码如下：

```
public class Theme {
    private int id;
    private String name;
    private String descr;
    //属性的 getter 和 setter 方法略
    ...
}
```

（4）创建数据库访问工具类。创建 DB 类，将对数据库操作的常用方法封装到此类中，并抛出访问数据库的异常，代码如下：

```
package com.cvit.struts2.util;
```

```
import java.sql.*;

public class DB {
    public static Connection createConn() throws SQLException,
    ClassNotFoundException {                //连接数据库
        Connection conn = null;
        Class.forName("com.mysql.jdbc.Driver");
        conn = DriverManager.getConnection("jdbc:mysql://localhost/BBS",
                "root", "root");
        return conn;
    }

    public static PreparedStatement prepare(Connection conn, String sql)
    throws SQLException {                   //执行 SQL 语句
        PreparedStatement ps = null;
        ps = conn.prepareStatement(sql);
        return ps;
    }

    //关闭连接
    public static void close(Connection conn) throws SQLException {
        conn.close();
        conn = null;
    }

    public static void close(Statement stmt) throws SQLException {
        stmt.close();
        stmt = null;
    }

    public static void close(ResultSet rs) throws SQLException {
        rs.close();
        rs = null;
    }
}
```

(5) 创建业务逻辑组件。创建 ThemeService 类，定义 list 方法，从数据库中读取所有主题信息，并抛出访问数据库异常，代码如下：

```
package com.cvit.struts2.sevice;
import java.sql.*;
import java.util.ArrayList;
import java.util.List;
import com.cvit.struts2.model.Theme;
import com.cvit.struts2.util.DB;

public class ThemeService {

    public List<Theme> list() throws SQLException,ClassNotFoundException {
        Connection conn = DB.createConn();
```

```
        String sql = "select * from _theme";
        PreparedStatement ps = DB.prepare(conn, sql);
        List<Theme> themes = new ArrayList<Theme>();

        ResultSet rs = ps.executeQuery();
        Theme c = null;
        while (rs.next()) {
            c = new Theme();
            c.setId(rs.getInt("id"));
            c.setName(rs.getString("name"));
            c.setDescr(rs.getString("descr"));
            themes.add(c);
        }
        DB.close(ps);
        DB.close(conn);
        return themes;
    }
}
```

(6) 创建 Action 类。创建 ThemeAction，定义 list 方法处理显示主题信息列表的请求，并抛出访问数据库异常，主要代码如下：

```
package com.cvit.struts2.action;
import java.sql.SQLException;
import java.util.List;
import com.cvit.struts2.model.Theme;
import com.cvit.struts2.sevice.ThemeService;
import com.opensymphony.xwork2.ActionSupport;
public class ThemeAction extends ActionSupport {
    private ThemeService themeService = new ThemeService();
    private List<Theme> themes;

    public String list() throws SQLException, ClassNotFoundException {
        themes = themeService.list();
        return SUCCESS;
    }
    //属性的 getter 和 setter 方法略
    ...
}
```

(7) 配置 Action 和访问数据库异常映射。打开配置文件 struts.xml，输入以下代码。

```
<package name="theme" extends="BBS_default" namespace="/theme">
    <action name="list" class="com.cvit.struts2.action.ThemeAction"
            method="list">
        <result>/Theme_list_success.jsp</result>
        <exception-mapping result="error" exception="java.sql.SQLException" />
        <result name="error">/error_sql.jsp</result>
    </action>
</package>
```

(8) 测试。部署工程，在浏览器的地址栏中输入 http://localhost:8080/BBS_Struts2_1600_DefaultInterceptor，打开 index.jsp 页面，不打开数据库，单击“主题列表”超链接，出现访问数据库异常，浏览器打开 error_sql.jsp，显示访问数据库异常信息，如图 18.4 所示。

(9) 配置全局映射和全局结果集。在公共包中将异常映射配置为全局映射，将结果配置为全局结果集，并配置 Action 所在包继承公共包，主要代码如下：

```
<package name="BBS_default" extends="struts-default">
    <global-results>
        <result name="error">/error_sql.jsp</result>
    </global-results>
    <global-exception-mappings>
        <exception-mapping result="error" exception="java.sql.SQLException" />
    </global-exception-mappings>
</package>
<package name="theme" extends="BBS_default" namespace="/theme">
    <action name="list" class="com.cvit.struts2.action.ThemeAction"
            method="list">
        <result>/Theme_list_success.jsp</result>
    </action>
</package>
```

(10) 测试。部署工程，在浏览器的地址栏中输入 http://localhost:8080/BBS_Struts2_1600_DefaultInterceptor，打开 index.jsp 页面，不打开数据库，单击“主题列表”超链接，出现访问数据库异常，浏览器打开 error_sql.jsp，显示访问数据库异常信息，如图 18.4 所示。

3. 任务总结

本任务以论坛管理系统中显示主题列表为例，使用 exception 拦截器实现声明式异常处理。实现方法是将访问数据库方法产生的异常层层上抛，直至抛给 Action，Action 将异常抛给 exception 拦截器，exception 拦截器根据 struts.xml 配置文件、根据异常的类型 java.sql.SQLException 映射逻辑视图 error，系统将逻辑视图对应的物理视图 error_sql.jsp 显示在浏览器中。

18.2.3 拓展任务 1：实现后台主题管理模块中声明式异常处理

1. 任务要求

实现后台主题管理模块中 ClassNotFoundException 异常处理的配置。

2. 建议步骤

(1) 使用工程 BBS_Struts2_1600_DefaultInterceptor。
(2) 创建视图层文件，用于显示 ClassNotFoundException 异常提示信息。
(3) 编写 struts.xml 文件，配置异常处理映射。
(4) 部署工程。
(5) 调试系统。打开浏览器，输入客户端请求，测试程序，如有错误，进行修改。

18.2.4　拓展任务 2：实现后台用户管理模块中声明式异常处理

1. 任务要求

采用声明的方式，实现用户模块中访问数据库异常和 ClassNotFoundException 异常的处理。

2. 建议步骤

(1) 使用工程 BBS_Struts2_1600_DefaultInterceptor。

(2) 创建视图层文件。

(3) 创建 User 类，定义属性。

(4) 创建 UserService 类，实现增、删、改、查业务逻辑。

(5) 创建 UserAction 类，实现处理增、删、改、查请求的方法。

(6) 编写 struts.xml 文件，配置 Action 和声明式异常处理。

(7) 打开视图层文件，编写提交请求、显示数据等代码。

(8) 部署工程。

(9) 调试系统。打开浏览器，输入客户端请求，测试程序，如有错误，进行修改。

18.2.5　Struts 2 非默认内建拦截器的使用

Struts 2 框架还有一些内建拦截器不默认执行，当需要执行这些拦截器时，才将其应用到系统中。

1. 拦截器 timer 的使用

timer 拦截器是 Struts 2 框架非默认执行的拦截器，用于显示 Action 的执行时间。

下面我们以拦截器 timer 为例，讲解如何将 Struts 2 非默认执行的内建拦截器应用到系统中。

(1) 首先创建 Action，代码如下：

```
public class ThemeAction extends ActionSupport {
    public String add(){
        System.out.println("执行 Action");
        return SUCCESS;
    }
}
```

(2) 在 struts.xml 中将 timer 拦截器配置到 Action 中，代码如下：

```
<package name="theme" extends="struts-default" namespace="/theme">
    <action name="add" class="com.cvit.struts2.action.ThemeAction" method="add">
        <result>/Theme_add_success.jsp</result>
        <interceptor-ref name="timer" />
    </action>
```

```
</package>
```

(3) 执行结果。发布运行程序后，提交请求，调用 ThemeAction 的 add()方法，timer 拦截器在控制台输出 Action 执行的时间为 51ms。

```
执行 Action
2013-1-26 14:25:46 com.opensymphony.xwork2.util.logging.jdk.JdkLogger info
信息: Executed action [/theme/add!add] took 51 ms.
```

2. 拦截器 token 的使用

token 拦截器也是 Struts 2 框架非默认执行的拦截器，用于防止表单重复提交。

例如，为论坛添加主题时，如果提交表单后刷新页面，会导致向数据库中插入多条记录，添加多个同名主题。使用 token 拦截器可以实现防止表单的重复提交，它的原理是：在页面插入<s:token/>标签，当提交表单时，服务器由 TokenHelper 类产生一个唯一的令牌键值对，并在提交表单时发送给服务器。服务器将客户端提交的令牌和缓存的令牌进行比较，如果是有效的，则清除服务器端缓存并继续处理，如果是无效的，则返回 invalid.token 结果。

下面我们以拦截器 token 为例，讲解将 Struts 2 非默认执行的内建拦截器应用到系统中的注意事项。

(1) 首先，在表单提交页面 Theme_add_input.jsp 插入<s:token/>标签，示例代码如下：

```
<body>
    添加主题
    <form action="theme/add" method="post">
        主 题 名: <input type="text" name="name"/><br/>
        主题简述: <input type="text" name="descr"/><br/>
        <input type="submit" value="添加"/>
        <s:token></s:token>
    </form>
</body>
```

(2) 创建 error.jsp 作为表单重复提交时显示提示信息的页面，代码如下所示。

```
<body>请不要重复提交</body>
```

(3) 创建 Theme_add_success.jsp，用于显示添加成功信息，代码如下：

```
<body>添加主题成功!</body>
```

(4) 创建 Action，代码如下：

```
public class ThemeAction extends ActionSupport {
    private String name;
    private String descr;
    public String add() throws Exception {
        System.out.println("向数据库中添加主题");
        System.out.println("主题名为: "+theme.getName());
```

```
            System.out.println("主题描述为: "+theme.getDescr());
            return SUCCESS;
            return SUCCESS;
        }
        //属性 getter 和 setter 方法略
    }
```

（5）在 struts. xml 中将 token 拦截器配置到 Action 中，代码如下：

```
<package name="theme" extends="struts-default" namespace="/theme">
    <action name="add" class="com.cvit.struts2.action.ThemeAction" method="add">
        <result>/Theme_add_success.jsp</result>
        <interceptor-ref name="token"></interceptor-ref>
        <result name="invalid.token">/error.jsp</result>
    </action>
</package>
```

因为重复提交时，服务器返回 invalid. token 结果，<result name="invalid. token">/error. jsp</result>配置“invalid. token”所对应的结果页面为 error. jsp。

（6）执行结果。发布运行程序后，打开 Theme_add_input. jsp 页面，添加表单并提交请求，调用 ThemeAction 的 add()方法，页面提示错误，如图 18. 5 所示。

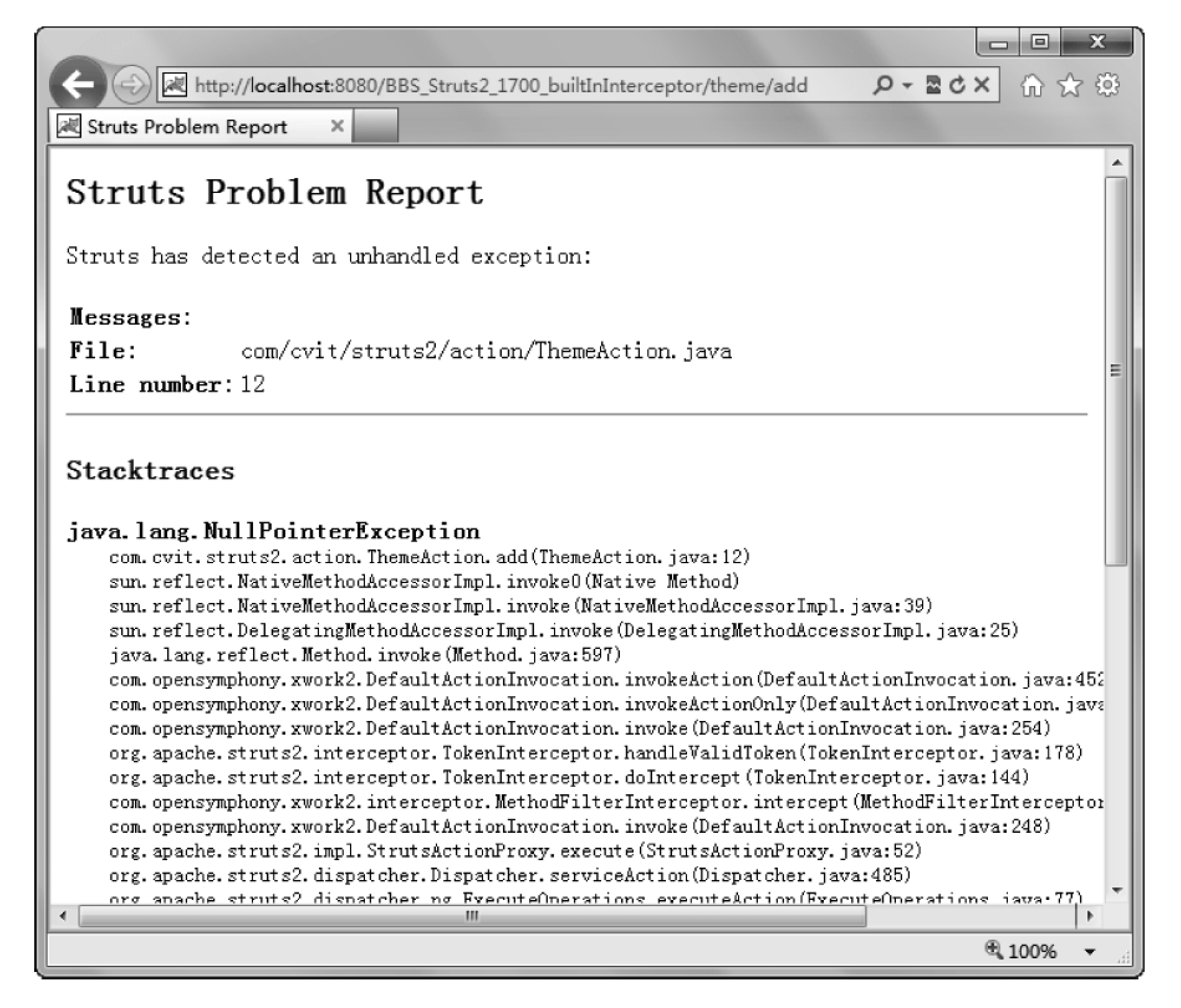


图 18. 5　错误提示信息

这是因为，当 Action 显式配置拦截器后，默认拦截器则失效。struts-default.xml 中 defaultStack 拦截器栈中的默认拦截器将不执行，页面提交的参数无法传递给 Action 引起的。

解决此问题的方法是，将 defaultStack 拦截器栈也配置给 Action，代码如下：

```
<package name="theme" extends="struts-default" namespace="/theme">
    <action name="add" class="com.cvit.struts2.action.ThemeAction" method="add">
        <result>/Theme_add_success.jsp</result>
        <interceptor-ref name="defaultStack"/>
        <interceptor-ref name="token"/>
        <result name="invalid.token">/error.jsp</result>
    </action>
</package>
```

重新发布运行程序后，打开 Theme_add_input.jsp 页面，添加表单并提交请求，调用 ThemeAction 的 add() 方法，显示 Theme_add_success.jsp 页面，刷新页面，显示 error.jsp 页面，提示不要重复提交表单，如图 18.6 所示。

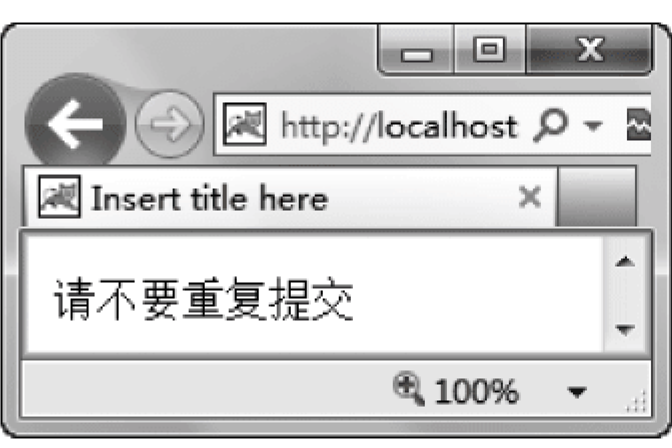


图 18.6　error.jsp 页面

18.2.6　实践任务 2：使用 token 拦截器实现防止表单重复提交

1. 任务说明

本任务实践使用 token 拦截器实现防止表单重复提交。

在论坛管理系统中，添加主题时，如果重复提交表单，页面显示如图 18.6 所示的提示信息。

2. 任务实施

(1) 创建工程。创建工程 BBS_Struts2_1700_builtInInterceptor。

(2) 视图层文件。

① 创建 Theme_add_input.jsp，插入<s:token/>标签，主要代码如下：

```
<body>
    添加主题
    <form action="theme/add" method="post">
        主 题 名：<input type="text" name="name"/><br/>
        主题简述：<input type="text" name="descr"/><br/>
        <input type="submit" value="添加"/>
        <s:token></s:token>
    </form>
</body>
```

② 创建 error.jsp，为表单重复提交时显示提示信息，代码如下：

```
<body>请不要重复提交</body>
```

③ 创建 Theme_add_success.jsp，用于显示添加成功信息，代码如下：

```
<body>添加主题成功!</body>
```

(3) 创建 Model 类。创建 Theme 类，代码如下：

```
public class Theme {
    private int id;
    private String name;
    private String descr;
    //属性的 getter 和 setter 方法略
    ...
}
```

(4) 创建 Action 类。创建 ThemeAction，定义 add 方法处理添加主题请求，主要代码如下：

```
package com.cvit.struts2.action;
import com.cvit.struts2.model.Theme;
import com.opensymphony.xwork2.ActionSupport;
public class ThemeAction extends ActionSupport {
    private Theme theme;
    public String add() {
        System.out.println("向数据库中添加主题");
        System.out.println("主题名为: "+theme.getName());
        System.out.println("主题描述为: "+theme.getDescr());
        return SUCCESS;
    }
    //属性的 getter 和 setter 方法略
    ...
}
```

(5) 配置 Action、token 拦截器和 invalid.token 对应的物理视图，并配置 defaultStack 拦截器栈。打开配置文件 struts.xml，输入以下代码。

```
<package name="theme" extends="struts-default" namespace="/theme">
    <action name="add" class="com.cvit.struts2.action.ThemeAction" method="add">
        <result>/Theme_add_success.jsp</result>
        <interceptor-ref name="defaultStack"/>
        <interceptor-ref name="token"></interceptor-ref>
        <result name="invalid.token">/error.jsp</result>
    </action>
</package>
```

(6) 测试。部署工程，在浏览器的地址栏中输入 http://localhost:8080/ BBS_Struts2_1700_builtInInterceptor/ Theme_add_input.jsp，打开 Theme_add_input.jsp 页面，输入主题名称和描述信息，单击“添加”按钮，页面跳转到 Theme_add_success.jsp 页面，刷新页面，浏览器打开 error.jsp，显示“不要重复提交”提示信息，如图 18.6 所示。

3. 任务总结

本任务以论坛管理系统中添加主题为例，使用 token 拦截器实现防止表单重复提交。

实现的方法是在表单提交页插入<s:token/>标签，在 Action 中配置 token 拦截器和"invalid. token"对应的物理视图。

需要注意的是，当为 Action 显示配置拦截器后，默认拦截器不执行，需要将 defaultStack 拦截器栈配置给 Action，而且要写在 token 拦截器配置前，让 token 拦截器最后执行。这是因为在 token 拦截器执行前，需要先执行参数传递、类型转换等操作。

18.2.7 拓展任务 3：实现防止重复提交修改主题的表单

1. 任务要求

使用 token 拦截器，实现后台主题管理模块中防止修改主题表单的重复提交。

2. 建议步骤

（1）使用工程 BBS_Struts2_1700_builtInInterceptor。

（2）创建视图层文件修改主题表单页面和修改主题成功信息页面，在修改主题表单页面插入<s:token>标签。

（3）在 Action 中添加 modify()方法，用于处理修改主题信息的请求。

（4）编写 struts. xml 文件，配置 Action 和 token 拦截器、defaultStack 拦截器栈，并配置 invalid. token 对应的物理视图为 error. jsp。

（5）部署工程。

（6）调试系统。打开浏览器，输入客户端请求，测试程序，如有错误，进行修改。

18.3 Struts 2 拦截器栈

在 18.2 节的例子中，为 Action 配置了 token 拦截器和 defaultStack 拦截器栈，在实际开发中，一个 Action 可能需要配置更多的拦截器，如果每个 Action 都单独配置拦截器，不仅代码臃肿，而且管理或维护都非常困难。使用拦截器栈可以轻松解决这个问题。

实际上我们可以把拦截器栈看做一个拦截器，只不过它是多个拦截器的集合，它在配置 Action 时与拦截器配置方式一致。

配置拦截器栈使用<interceptor-stack>标签，<interceptor-stack>标签必须写在<interceptors>元素内，格式如下：

```
<interceptors>
    <interceptor-stack name="拦截器栈名">
        <interceptor-ref name="拦截器 1"/>
        <interceptor-ref name="拦截器 2"/>
        <interceptor-ref name="拦截器 3"/>
        ⋮
        <interceptor-ref name="拦截器 n"/>
    </interceptor-stack>
</interceptors>
```

其中拦截器 n 可以是一个拦截器，也可以是一个拦截器栈。例如，把前面讲解的 timer 拦截器、token 拦截器和 defaultStack 拦截器栈配置为一个拦截器栈，代码如下：

```
<interceptors>
    ...
    <interceptor-stack name="myDefaultStack">
        <interceptor-ref name="defaultStack"/>
        <interceptor-ref name="timer"/>
        <interceptor-ref name="token"/>
    </interceptor-stack>
    ...
</interceptors>
```

为 Action 配置此拦截器栈与配置拦截器方式一致，代码为<interceptor-ref name="myDefaultStack"/>。

18.4　配置默认拦截器

Struts 2 框架将拦截器栈 defaultStack 配置为默认拦截器。我们也可以将自己定义的拦截器栈或拦截器(如上节中拦截器栈 myDefaultStack)配置为默认拦截器。

配置默认拦截器的方法是，在一个包中使用<default-interceptor-ref name="拦截器名"/>指定默认拦截器，则此包中所有没有显式指定拦截器的 Action 都默认执行此拦截器，当其他包中的 Action 想应用此默认拦截器，只需要继承此包即可。

在实际开发中，一般是将默认拦截器等公共配置配置在一个公共包中，其他包中的 Action 要使用默认拦截器，继承此公共包即可。

下面是示例代码。

```
<package name="default" extends="struts-default" namespace="">
<interceptors>
<!--配置拦截器栈-->
    <interceptor-stack name="myDefaultStack">
        <interceptor-ref name="defaultStack" />
        <interceptor-ref name="timer" />
        <interceptor-ref name="token" />
    </interceptor-stack>
</interceptors>
<!--配置默认拦截器-->
<default-interceptor-ref name="myDefaultStack" />
<!--配置全局结果-->
<global-results>
    <result name="invalid.token">/error.jsp</result>
</global-results>
</package>
<package name="theme" extends="default" namespace="/theme">
    <action name="add" class="com.cvit.struts2.action.ThemeAction" method="add">
        <result>/Theme_add_success.jsp</result>
```

```
    </action>
</package>
```

default 包中配置了全局结果 invalid. token、拦截器栈 myDefaultStack，并将 myDefaultStack 配置为默认拦截器。

theme 包继承 default 包，则 theme 包中的所有 Action 都应用默认拦截器和全局结果集。

18.5 实践任务 3：配置拦截器栈和默认拦截器

1. 任务说明

本任务实践配置拦截器栈和默认拦截器。

在论坛管理系统的主题管理模块中，将 timer 拦截器、token 拦截器和 defaultStack 拦截器栈配置为 myDefaultStack，并将 myDefaultStack 配置为默认拦截器。

2. 任务实施

(1) 创建工程。打开工程 BBS_Struts2_1700_builtInInterceptor，复制并粘贴，重命名为 BBS_Struts2_1750_defaultStack。

(2) 配置 Action、拦截器栈、默认拦截器和全局结果。打开配置文件 struts. xml，输入以下代码。

```
<package name="default" extends="struts-default" namespace="">
    <interceptors>
    <!--配置拦截器栈-->
    <interceptor-stack name="myDefaultStack">
        <interceptor-ref name="defaultStack" />
        <interceptor-ref name="timer" />
        <interceptor-ref name="token" />
    </interceptor-stack>
    </interceptors>
    <!--配置默认拦截器-->
    <default-interceptor-ref name="myDefaultStack" />
    <!--配置全局结果-->
    <global-results>
        <result name="invalid.token">/error.jsp</result>
    </global-results>
</package>
<package name="theme" extends="default" namespace="/theme">
    <action name="add" class="com.cvit.struts2.action.ThemeAction" method="add">
        <result>/Theme_add_success.jsp</result>
    </action>
</package>
```

(3) 测试。部署工程，在浏览器的地址栏中输入 http://localhost:8080/BBS_Struts2_1750_defaultStack/Theme_add_input.jsp，打开 Theme_add_input.jsp 页面，输入主题名称和描述信息，单击“添加”按钮，页面跳转到 Theme_add_success.jsp 页面，刷新页面，浏览器打开 error.jsp，显示“不要重复提交”提示信息，如图 18.6 所示，控制台输

出以下信息。

```
向数据库中添加主题
主题名为：aaa
主题描述为：123
2013-1-26 16:49:08
com.opensymphony.xwork2.util.logging.jdk.JdkLogger info
信息：Executed action [/theme/add!add] took 98 ms.
2013-1-26 16:49:15
com.opensymphony.xwork2.util.logging.jdk.JdkLogger warn
警告：Form token 3G02Z04RQOAM5EES9Q114DFE3M981RM7 does not match the session token null.
2013-1-26 16:49:15
com.opensymphony.xwork2.util.logging.jdk.JdkLogger info
信息：Executed action [/theme/add!add] took 107 ms.
```

3. 任务总结

本任务以论坛管理系统中添加主题为例，配置拦截器栈和默认拦截器。

18.6　自定义拦截器

Struts 2 框架提供了丰富的内建拦截器，可以实现 Web 应用中的大部分通用功能。当 Struts 2 内建拦截器不能满足需求时，我们还可以自定义拦截器，以实现功能。

下面讲解如何实现拦截器类、如何部署拦截器、如何为 Action 添加拦截器配置，同时也帮助我们更加清晰地理解 Struts 2 内建拦截器是如何运作的。

18.6.1　定义拦截器类

拦截器类可以通过实现 Interceptor 接口的方式定义。

Interceptor 接口的主要代码如下：

```
public interface Interceptor extends Serializable {
    void destroy();
    void init();
    String intercept(ActionInvocation invocation) throws Exception;
}
```

Interceptor 接口共有三个抽象方法：init()、intercept()、destroy()。

(1) init()：只在加载拦截器时执行一次，是拦截器的初始化方法，用于加载所需资源等。

(2) intercept()：该方法是实现拦截功能代码。

(3) destroy()：只在拦截器被销毁前执行一次，释放 init()方法加载的资源等。

例如，定义 MyFirstInterceptor 类实现 Interceptor 接口，代码如下：

```
package com.cvit.struts2.interceptor;

import com.opensymphony.xwork2.ActionInvocation;
```

```
import com.opensymphony.xwork2.interceptor.Interceptor;

public class MyFirstInterceptor implements Interceptor {

    @Override
    public void destroy() {
        //销毁方法
    }

    @Override
    public void init() {
        //初始化方法
    }

    @Override
    public String intercept(ActionInvocation invocation) throws Exception {
        //拦截方法
        System.out.println("拦截器代码段 1");
        String result=invocation.invoke();
        System.out.println("拦截器代码段 2");
        return result;
    }
}
```

18.6.2 部署拦截器

自定义拦截器需要使用<interceptor>标签，<interceptor>标签与<interceptor-stack>标签同样，必须写在<interceptors>元素内，格式如下：

```
<interceptor name="拦截器名" class="拦截器类"/>
```

如 18.6.1 小节中定义的拦截器类，部署代码如下：

```
<interceptor name="MyFirstInterceptor" class="com.cvit.struts2.interceptor.MyFirstInterceptor"/>
```

18.6.3 应用拦截器

将自定义拦截器应用给 Action 的方式，与内建拦截器相同。例如，MyFirstInterceptor 拦截器配置 Action 的代码如下：

```
<interceptor-ref name="MyFirstInterceptor"/>
```

当 Action 被请求时，在 Action 执行前，执行 MyFirstInterceptor 拦截器的“拦截器代码段 1”，然后执行 Action，Action 执行后，执行 MyFirstInterceptor 拦截器的“拦截器代码段 2”。

18.6.4　实践任务 4：自定义拦截器

1. 任务说明

本任务实践自定义拦截器的实现与应用。

在论坛管理系统的主题管理模块中，自定义拦截器 MyFirstInterceptor，并部署拦截器，应用给 Action，最后测试拦截器。

2. 任务实施

（1）创建工程。创建工程 BBS_Struts2_1800_myInterceptor。

（2）视图层文件。创建 Theme_list_success.jsp，模拟显示主题列表。

```
<body>
    主题列表<br/>
    主题列表<br/>
    主题 1<br/>
    主题 2<br/>
    主题 3<br/>
    <a href="quit">退出系统</a>
</body>
```

（3）创建拦截器。

定义 MyFirstInterceptor 类实现 Interceptor 接口，代码如下：

```
package com.cvit.struts2.interceptor;

import com.opensymphony.xwork2.ActionInvocation;
import com.opensymphony.xwork2.interceptor.Interceptor;

public class MyFirstInterceptor implements Interceptor {

    @Override
    public void destroy() {
        //销毁方法
    }

    @Override
    public void init() {
        //初始化方法
    }

    @Override
    public String intercept(ActionInvocation invocation) throws Exception {
        //拦截方法
        System.out.println("拦截器代码段 1");
        String result=invocation.invoke();
        System.out.println("拦截器代码段 2");
```

```
            return result;
        }
    }
```

(4) 创建 Action 类。创建 ThemeAction,定义 list 方法模拟列表主题信息,主要代码如下:

```
package com.cvit.struts2.action;
import com.opensymphony.xwork2.ActionSupport;
public class ThemeAction extends ActionSupport {
    public String list() {
        System.out.println("访问数据库,显示用户列表");
            return SUCCESS;
    }
}
```

(5) 部署拦截器。打开配置文件 struts.xml,输入以下代码。

```
<package name="myDefault" extends="struts-default">
    <interceptors>
        <interceptor name="MyInterceptor"
            class="com.cvit.struts2.interceptor.MyFirstInterceptor"/>
    </interceptors>
</package>
```

(6) 配置 Action,应用拦截器。打开配置文件 struts.xml,输入以下代码。

```
<package name="main" extends="myDefault" namespace="/">
    <action name="list" class="com.cvit.struts2.action.ThemeAction" method="list">
        <result>/Theme_list_success.jsp</result>
        <interceptor-ref name="MyInterceptor" />
    </action>
</package>
```

(7) 测试。部署工程,在浏览器的地址栏中输入 http://localhost:8080/BBS_Struts2_1800_myInterceptor/list,打开 Theme_list_success.jsp 页面,控制台输出信息如下:

```
拦截器代码段 1
访问数据库,显示用户列表
拦截器代码段 2
```

(8) 取消拦截器。如果<action name="list"…/>不想应用拦截器 MyInterceptor,只须删除代码<interceptor-ref name="MyInterceptor" />。

将 Action 配置代码作如下修改。

```
<action name="list" class="com.cvit.struts2.action.ThemeAction" method="list">
    <result>/Theme_list_success.jsp</result>
</action>
```

(9) 测试。部署工程,在浏览器的地址栏中输入 http://localhost:8080/BBS_

Struts2_1800_myInterceptor/list，打开 Theme_list_success.jsp 页面，控制台输出信息如下：

```
访问数据库，显示用户列表
```

3. 任务总结

本任务以论坛管理系统中添加主题为例，实践自定义拦截器，部署和应用拦截器。通过自定义拦截器，可以观察拦截器代码与 Action 代码执行顺序。

本任务还实践了拦截器方便地插拔方式，拦截器类与 Action 类代码可以看作无耦合，拦截功能通过配置代码实现。

18.6.5　拓展任务 4：应用自定义拦截器

1. 任务要求

为 Action 应用自定义拦截器 MyInterceptor。

2. 建议步骤

(1) 使用工程 BBS_Struts2_1800_myInterceptor。
(2) 创建视图层文件 login.jsp。
(3) 在 Action 中添加 quit()方法，用于退出系统。
(4) 编写 struts.xml 文件，配置 Action 退出系统，并应用拦截器 MyInterceptor。
(5) 部署工程。
(6) 调试系统。打开浏览器，输入客户端请求，测试程序，如有错误，进行修改。

18.6.6　自定义权限验证拦截器

几乎每个 Web 应用都需要权限验证，当用户提交请求时，首先查看用户是否有权限进行此操作，如果有则继续执行，如果没有则返回提示信息。

例如，用户提交显示主题列表请求，使用权限验证拦截器拦截请求，查看用户是否登录，如果登录，则显示主题列表页面，如果未登录，则返回登录页面，并提示用户登录系统。

权限验证拦截器实现的原理一般如下：

用户登录系统后，通常会放一个值(例如用户名)在 session 中，权限验证拦截器查看 session 中是否有这个值，有则继续执行 Action，没有则返回登录页面，并提示用户登录系统再进行操作。

权限验证拦截器的实现代码如下：

```java
package com.cvit.struts2.interceptor;
import java.util.Map;
import com.opensymphony.xwork2.ActionInvocation;
import com.opensymphony.xwork2.interceptor.AbstractInterceptor;
public class AuthorityInterceptor extends AbstractInterceptor{
```

```
    @Override
    public String intercept(ActionInvocation invocation) throws Exception {
        Map session=invocation.getInvocationContext().getSession();//获得 session
        String user=(String) session.get("user");             //从 session 中获得 user
        if(user!=null){                            //如果 user 不为空,则用户已经登录,继续执行
            return invocation.invoke();
        }
        //否则将提示信息放到 session 中,并返回"login"逻辑视图,不执行 Action
        session.put("tip", "您还没有登录系统,请输入用户名和密码登录");
        return "login";
    }
}
```

拦截器类如果不需要初始化一些资源,可以直接继承 AbstractInterceptor 类,重写 intercept()方法即可。

18.6.7 实践任务 5:自定义权限验证拦截器

1. 任务说明

本任务实践自定义权限验证拦截器的实现与应用。

在论坛管理系统中,自定义权限验证拦截器 AbstractInterceptor,实现登录权限验证。

2. 任务实施

(1) 打开工程。打开工程 BBS_Struts2_1800_myInterceptor。

(2) 视图层文件。

① 创建 login.jsp,代码如下:

```
<body>
    登录<br/>
    <s:property value="errors.loginFail[0]"/>
    <s:property value="#session.tip"/>
    <form action="login" method="post">
        用户名: <input type="text" name="name"></input><br/>
        密  码: <input type="text" name="password"></input><br/>
        <input type="submit" value="登录"/>
    </form>
</body>
```

② 创建 index.jsp 页面,代码如下:

```
<body>
    欢迎<s:property value="name"/>登录本系统!<br/>
    <a href="list">显示主题列表</a>
</body>
```

(3) 创建 Action 类。创建 UserAction,实现用户登录和退出系统,代码如下:

```
package com.cvit.struts2.action;
import java.util.Map;
import com.opensymphony.xwork2.ActionContext;
import com.opensymphony.xwork2.ActionSupport;
public class UserAction extends ActionSupport {
    private String name=null;
    private String password=null;
    //登录系统方法
    public String login() {
    //若输入用户名和密码为“aaa”、“123”,则将用户名赋给 user,放在 session 中
        if(name.equals("aaa")&&password.equals("123")){
            ActionContext ctx = ActionContext.getContext();
            Map session = ctx.getSession();
            session.put("user",getName());
            return SUCCESS;
        }
        else {//否则将提示信息放在 loginFail 中,并将 loginFail 放在 FieldError 中
            this.addFieldError("loginFail", "您输入的用户名或密码不正确,请重新输入");
            return "fail";
        }
    }
    //退出系统方法
    public String quit(){
        ActionContext ctx = ActionContext.getContext();
        Map session = ctx.getSession();
        session.clear();                              //清空 session 中的值
        return SUCCESS;
    }
    //属性的 getter 和 setter 方法略
}
```

(4) 创建拦截器。定义 AuthorityInterceptor 类继承 AbstractInterceptor 类，重写 intercept 方法，代码如下：

```
package com.cvit.struts2.interceptor;
import java.util.Map;
import com.opensymphony.xwork2.ActionInvocation;
import com.opensymphony.xwork2.interceptor.AbstractInterceptor;
public class AuthorityInterceptor extends AbstractInterceptor{
    @Override
    public String intercept(ActionInvocation invocation) throws Exception {
        Map session=invocation.getInvocationContext().getSession();     //获得 session
        String user=(String) session.get("user");          //从 session 中获得 user
        if(user!=null){                       //如果 user 不为空,则用户已经登录,继续执行
            return invocation.invoke();
        }
        //否则将提示信息放到 session 中,并返回“login”逻辑视图,不执行 Action
        session.put("tip", "您还没有登录系统,请输入用户名和密码登录");
        return "login";
```

```
        }
    }
```

(5) 部署拦截器。打开配置文件 struts.xml，部署拦截器 AuthorityInterceptor，并配置拦截器栈。另外，因为配置全局结果 login，代码如下：

```
<package name="myDefault" extends="struts-default">
    <interceptors>
        <interceptor name="AuthorityInterceptor"
                    class="com.cvit.struts2.interceptor.AuthorityInterceptor"/>
        ...
        <interceptor-stack name="mydefault">
            <interceptor-ref name="defaultStack" />
            <interceptor-ref name="AuthorityInterceptor"/>
        </interceptor-stack>
    </interceptors>
        <global-results>
            <!-- 当返回 login 逻辑视图时，转入/login.jsp 页面 -->
            <result name="login">/login.jsp</result>
        </global-results>
</package>
```

(6) 配置 Action，应用拦截器。打开配置文件 struts.xml，输入以下代码。

```
<package name="main" extends="myDefault" namespace="/">
    <action name="login" class="com.cvit.struts2.action.UserAction" method="login">
        <result>/index.jsp</result>
        <result name="fail">/login.jsp</result>
    </action>

    <action name="list" class="com.cvit.struts2.action.ThemeAction" method="list">
        <result>/Theme_list_success.jsp</result>
        <interceptor-ref name="mydefault"/>
    </action>
    <!-- 退出系统 -->
    <action name="quit" class="com.cvit.struts2.action.UserAction" method="quit">
        <result>/login.jsp</result>
        <interceptor-ref name="mydefault"/>
    </action>
</package>
```

(7) 测试。部署工程，在浏览器的地址栏中输入 http://localhost:8080/BBS_Struts2_1800_myInterceptor/list，跳转到 login.jsp 页面，并显示请登录系统提示信息，如图 18.7 所示。

3. 任务总结

本任务以论坛管理系统为例，通过自定义拦截器类，实现权限验证。

注意：权限验证拦截器不要拦截 login 请求，因为非登录用户是有登录权限的。

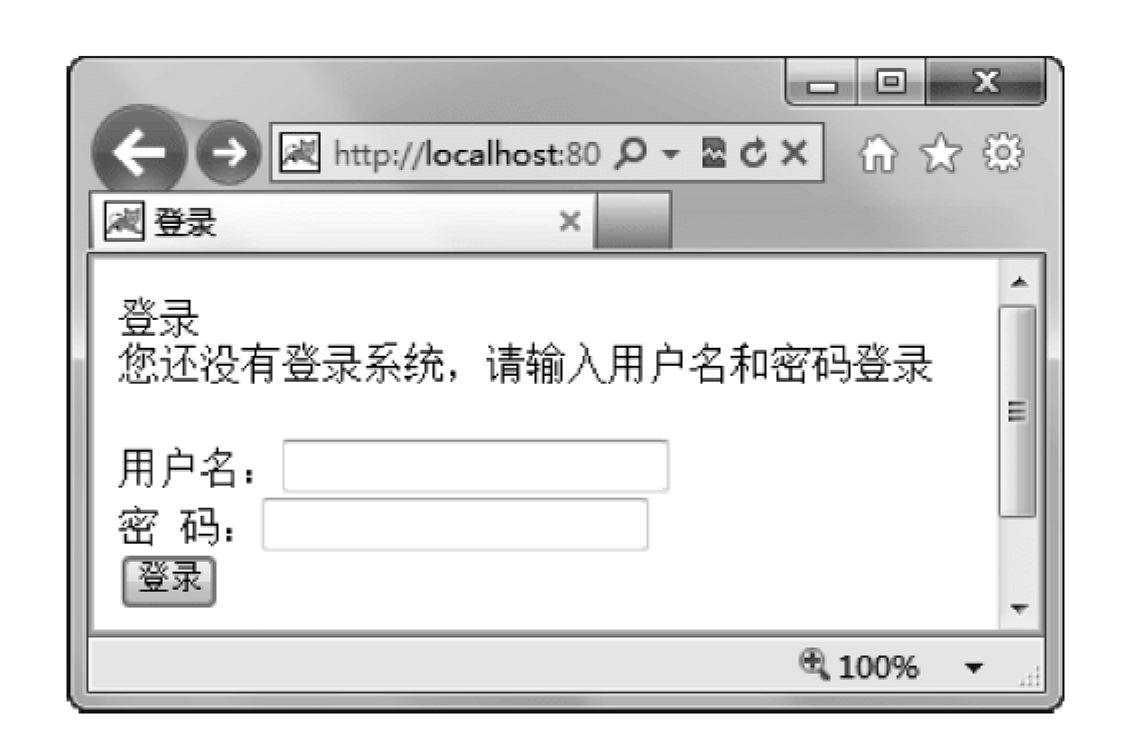


图 18.7　权限验证提示信息

18.6.8　拓展任务 5：实现主题管理模块的权限验证

1. 任务要求

使用自定义拦截器 AbstractInterceptor，实现主题管理模块中增、删、改、查操作的权限验证。

2. 建议步骤

(1) 使用工程 BBS_Struts2_1800_myInterceptor。
(2) 创建视图层文件。
(3) 在 Action 中添加方法，用于处理增、删、改、查主题请求。
(4) 编写 struts.xml 文件，配置 Action 和 AbstractInterceptor 拦截器。
(5) 部署工程。
(6) 调试系统：打开浏览器，输入客户端请求，测试程序，如有错误，进行修改。

18.7　实训：实现论坛管理系统的用户管理模块的权限验证

实训要求：
(1) 实现用户的登录、增、删、改、查等操作。
(2) 根据实际需求，为 Action 应用权限验证拦截器。
(3) 测试。

18.8　本章小结

本章首先介绍了使用拦截器的意义，接着讲解了什么是拦截器，它的设计机制是什么样的，使读者对拦截器有初步的认识。然后，从 Struts 2 内建拦截器的使用方法讲起，举例说明了默认内建拦截器和非默认内建拦截器的使用方法，讲解了拦截器栈和默认拦截器的作用和配置方法。最后，以自定义拦截器为例，讲解了拦截器的定义、部署和应用的方法。

拦截器是 Struts 2 框架的核心内容，掌握好拦截器可以开发优秀的系统。

第19章

基于 Struts 2 实现论坛管理系统

经过前面的学习,我们对 Struts 2 框架的技术已经有了较全面的掌握。本章以开发论坛管理系统为例,学习综合应用 Struts 2 框架技术开发系统。

本章要点:

- 系统分析与设计
- 规范设计的作用和内容
- 系统架构的搭建
- Struts 2 框架的 MVC 各层的作用
- Struts 2 框架的 MVC 各层的实现
- 实现数据输入的校验
- 配置拦截器

19.1 系统分析与设计

在开发系统之前,首先要进行需求分析和系统设计。

19.1.1 需求分析

论坛管理系统分为前台和后台两部分。

(1) 前台供用户使用,游客可以注册为会员,会员可以登录前台,阅读、发表、修改、查找、删除帖子,查看主题,修改本人的会员信息等。

(2) 后台供管理员使用,管理员可以登录后台,可以增加、删除、修改、查看会员,可以增加、删除、修改、查看主题,增加、删除、修改、查看帖子等。

19.1.2 功能设计

以论坛管理系统后台为例,划分为管理员登录、用户管理、主题管理、帖子管理四个模块,其中用户管理模块包括显示会员列表、增加会员、修改会员信息、删除会员功能,其后台管理用例如图 19.1 所示。

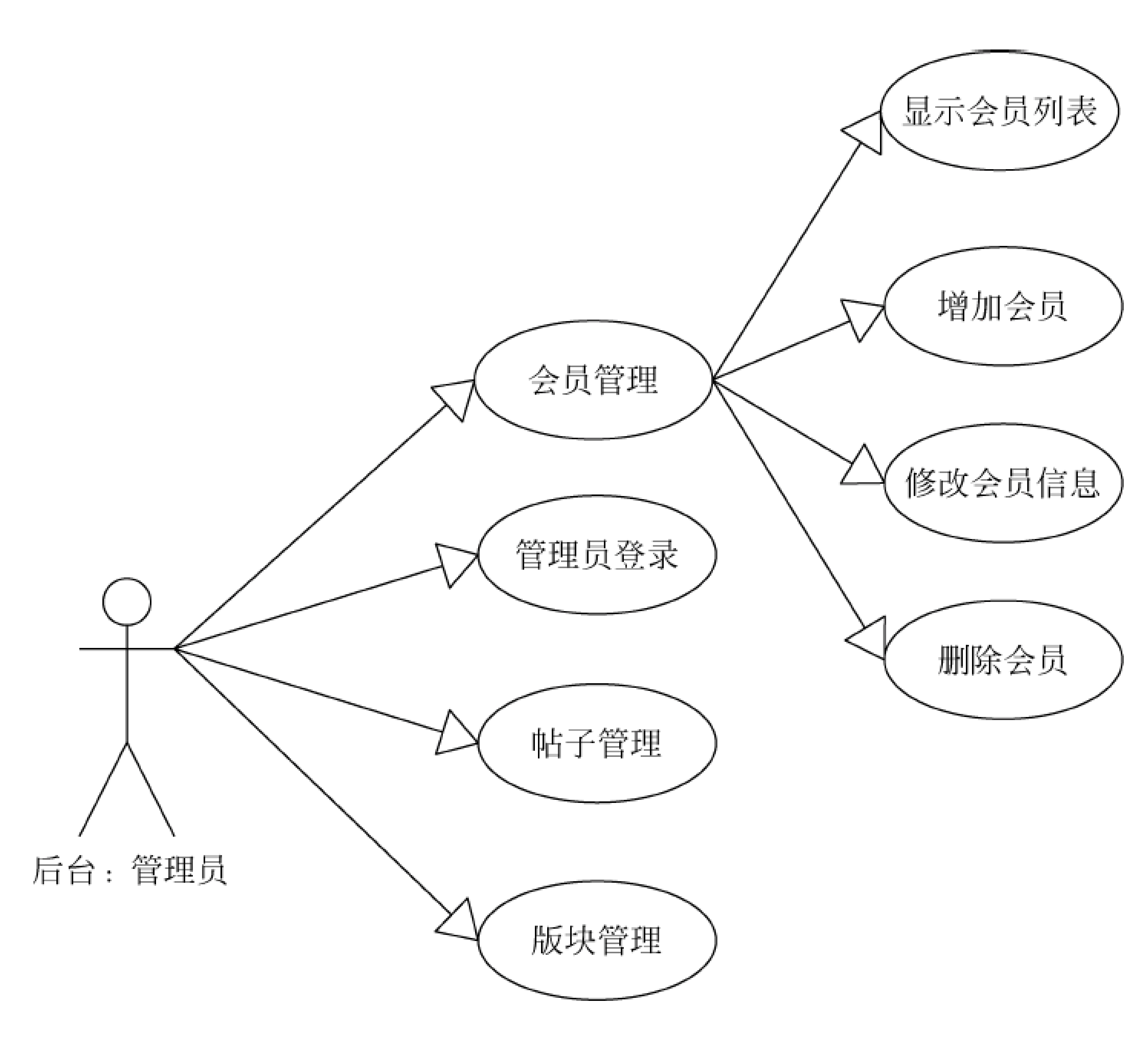


图 19.1　后台管理用例图

19.1.3　数据库设计

论坛管理系统数据库包括管理员表、会员表、主题表、帖子表，数据字典如表 19.1～表 19.4 所示。

表 19.1　管理员_admin

字段名	名　称	类　型	是否为空	备　注
id	编号	int	否	主键，自增
name	管理员名	varchar	否	
password	密码	varchar	否	

表 19.2　会员_user

字段名	名　称	类　型	是否为空	备　注
id	编号	int	否	主键，自增
name	会员名	varchar	否	3～8 位字母
password	密码	varchar	否	6～10 位字母
age	年龄	int		0～100

表 19.3 主题_theme

字段名	名　称	类　型	是否为空	备　注
id	编号	int	否	主键，自增
name	主题名	varchar	否	
descr	主题描述	varchar	否	

表 19.4 帖子_note

字段名	名　称	类　型	是否为空	备　注
id	编号	int	否	主键，自增
title	帖子题目	varchar	否	
content	帖子内容	varchar	否	
userId	发帖人	int	否	
themeId	所属主题	int	否	

19.2 命名规范设计的作用和内容

在需求分析和系统设计之后，开发系统之前，进行命名规范设计可以使开发更加规范，方便使用通配符配置 Action，方便分工合作开发系统，并提高系统的可读性、可维护性与可拓展性。

规范设计的内容主要包括以下几个方面。

① 包名命名规范。

② Java 类命名规范。

③ JSP 文件命名规范。

④ 数据库名、表名、字段名命名规范。

下面为论坛管理系统设计命名规范如下。

1. 包名命名规范

包名命名规范如下：

```
com.公司名.项目名.分层名
```

分层名称如下。

(1) util：关于数据库操作的工具类在此包中创建。

(2) Action：Struts 2 框架的 Action 类在此包中创建。

(3) Model：模型类在此包中创建。

(4) service：实现业务逻辑的类在此包中创建。

(5) VO：关于数据传递所使用的类在此包中创建。

2. Java 类命名规范

(1) Action 类

Action 类的命名规范包括类名和方法名的命名规范。

① 类名：模块名＋_＋Action，例如 User_Action、Theme_Action 等。

② 方法名：增、删、改、列表、查找方法统一命名为 add、delete、modify、list、query。

(2) Model 类

Model 类名规范包括类名与属性名的命名规范。

① 类名：与表名相同，如表名为_user，类名为 User。

② 属性名：与字段名相同。

(3) 业务逻辑类

① 类名：模块名＋Service。

② 方法名：命名规则与 Action 类方法名的命名规则一致。

3. JSP 文件命名规范

JSP 命令规范为"模块名_方法名_结果.jsp"。例如，显示用户列表成功页属于用户模块、调用 list 方法，结果是成功，文件名为 User_list_success.jsp；添加用户输入页属于用户模块，将调用 add 方法，属于输入页面，文件名为 User_add_input.jsp；数据库访问异常提示页面，属于所有模块，因此不指定模块名，大部分方法都可以引起数据库访问异常，因此不指定方法名，结果是异常，也就是出现了错误，文件名为 slq_error.jsp。

4. 数据库、表名、字段名命名规范

(1) 数据库名与系统名一致，命名为 DB_BBS。

(2) 表名要见名识义，并为避免与数据库关键字重复，表名以下划线开头，如_user 表示会员表。

(3) 字段名也要见名识义，可以采用适当缩写形式，并注意避免与数据库关键字相同。

19.3　系统架构的搭建

在软件开发过程中，一个具有最简功能的系统架构要尽早搭建起来，就像盖房子先将架子搭好，然后再筑墙、加窗、立门，最后搬进家具。

Struts 2 应用系统的架构，主要包括 JSP 文件和 Action 类，以实现请求的处理、页面之间的正确跳转。

因为整个论坛管理系统的架构内容较多，而且各模块之间也比较相似，所以我们以后台用户管理模块和管理员登录为例，讲解系统架构的搭建。

后台用户管理模块和管理员登录的架构图如图 19.2 所示。

图 19.2 中，Class-×××表示需要开发人员创建 Action 类，定义×××方法，处理页面提交的请求。

搭建系统架构步骤如下。

(1) 创建 Struts 2 应用。

(2) 创建视图层文件。

(3) 创建 Action。

(4) 在 struts. mxl 文件中配置 Action。

(5) 根据 Action 的配置,编写视图层文件的请求。

(6) 测试架构。

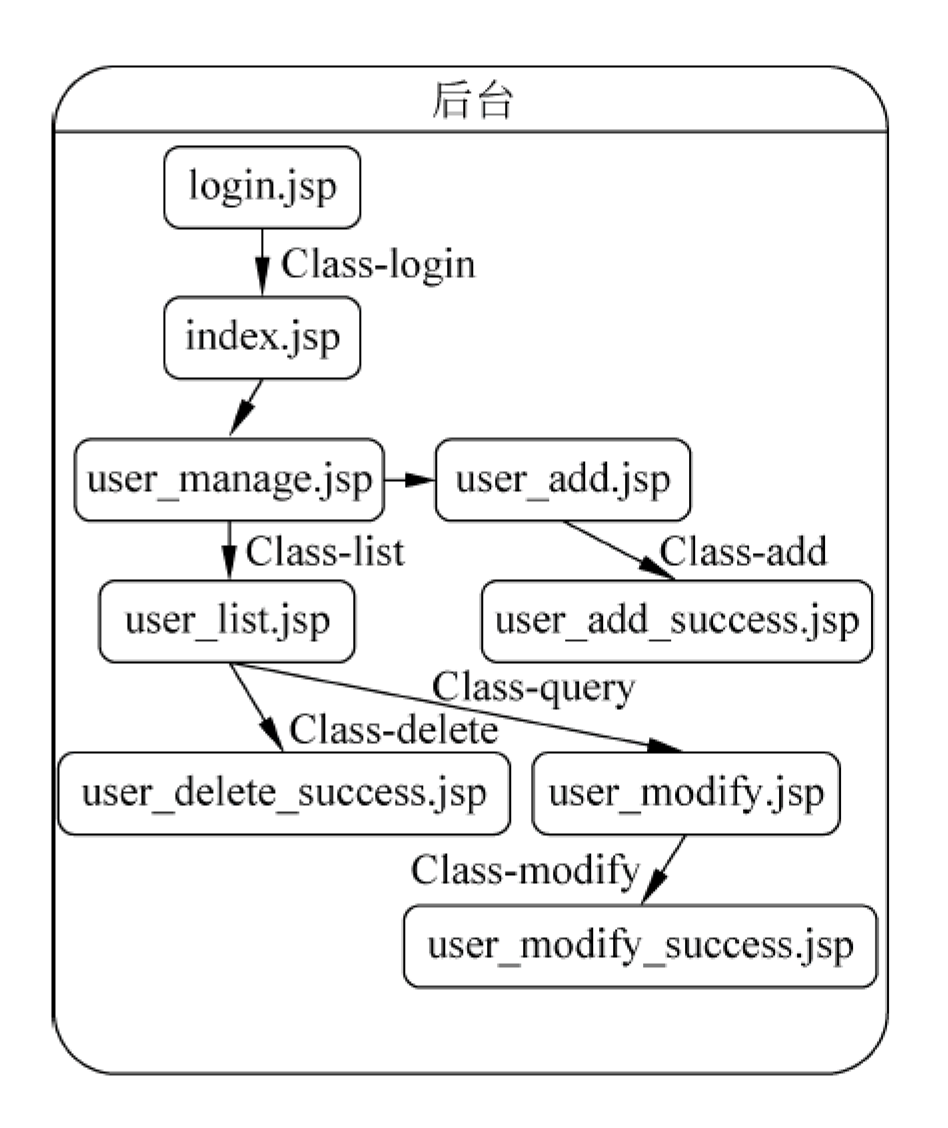


图 19.2 架构图

19.3.1 创建 Struts 应用

创建 Web 应用 BBS_Struts2_2000_Framework,导入 Struts 2 框架常用 JAR 包,添加 struts. xml 配置文件,并在 web. xml 中配置 Struts 2 的核心控制器 FilterDispatcher。

注意:本应用将操作数据库,因此关于数据库访问的 JAR 包也要导入系统中。

论坛管理系统采用 MySQL 数据库,因此导入 MySQL 的数据库访问 JAR 包,例如 mysql-connector-java-3. 1. 13-bin. jar。

19.3.2 创建视图层 JSP 文件

根据图 19.2 所示,创建 JSP 文件,并根据 JSP 文件功能,插入静态元素的代码。

1. login. jsp

login. jsp 是后台登录页面,用于管理员输入用户名和密码,提交登录请求。其主要代码如下:

```
<body>
    <form action="" method="post">
        <table align="center">
            <caption><h3>用户登录</h3></caption>
            <tr> <td>用户名: <input type="text" name=""/></td></tr>
            <tr> <td>密码: <input type="password" name=""/></td></tr>
            <tr align="center"><td>
                <input type="submit" value="登录"/>
                <input type="reset" value="重填" /></td>
            </tr>
        </table>
    </form>
</body>
```

2. index. jsp

index. jsp 为后台主页,主要代码如下:

```
<body>
    欢迎 *** 登录本系统<br/>
    <a href="">会员管理</a><br/>
    <a href="">帖子管理</a><br/>
    <a href="">主题管理</a><br/>
</body>
```

3. user_manage.jsp

user_manage.jsp 为用户管理模块主页，主要代码如下：

```
<body>
    <a href="">添加会员</a><br />
    <a href="">显示会员列表</a><br />
</body>
```

4. user_add_input.jsp

user_add_input.jsp 为添加用户页，主要代码如下：

```
<body>
<form action="" method="post">
<table align="center">
    <caption><h3>输入用户信息</h3></caption>
    <tr> <td>用户名: <input type="text" name=""/></td></tr>
    <tr> <td>密码: <input type="password" name=""/></td></tr>
    <tr> <td>确认密码: <input type="password" name=""/></td></tr>
    <tr> <td>年龄: <input type="text" name=""/></td></tr>
    <tr align="center">
        <td><input type="submit" value="添加"/>
            <input type="reset" value="重填" /></td>
    </tr>
</table>
</form>
</body>
```

5. user_list_success.jsp

user_list_success.jsp 为显示用户列表页，主要代码如下：

```
<body>
    会员列表<br/>
    会员 1 信息<a href="">修改会员</a>|<a href=""> 删除会员</a><br/>
</body>
```

鉴于篇幅所限，其他 JSP 页代码省略，请读者参照以上内容设计。

19.3.3 创建 Action 类

根据图 19.2 所示，创建 Action 类，定义方法用于处理请求。

1. 创建 Admin_Action

Admin_Action 用于处理管理员的登录请求，处理请求的方法是 login()方法，主要代

码如下：

```
package com.cvit.BBS.action;
import com.opensymphony.xwork2.ActionSupport;
public class Admin_Action extends ActionSupport {
    public String login() {
        System.out.println("------login------");    //用于测试方法调用的是否正确
        return SUCCESS;
    }
}
```

2. 创建 User_Action

User_Action 用于处理会员管理模块的请求，主要代码如下：

```
package com.cvit.BBS.action;
import com.opensymphony.xwork2.ActionSupport;
public class User_Action extends ActionSupport {
    //添加
    public String add() {
        return SUCCESS;
    }
    //删除
    public String delete() {
        return SUCCESS;
    }
    //修改
    public String modify() {
        return SUCCESS;
    }
    //列表
    public String list() {
        return SUCCESS;
    }
    //查询
    public String queryById() {
        return SUCCESS;
    }
}
```

也可以在方法中插入输出语句来帮助测试方法调用是否正确，或者在页面插入<s:debug/>标签以查看 Action 执行情况。

19.3.4 配置 Action

在实际开发系统时，开发组成员需要分工合作，如果每个开发组成员都将配置代码写在 struts.xml 文件中，势必为分工合作带来不便。Struts 2 框架允许编写多个配置文件，解决了这一问题。

在 Struts 应用中编写多个配置文件，其配置文件格式与 struts.xml 文件一致，然后

使用 include 标签将多个配置文件包含到 struts.xml 文件中，其效果等同于所有配置代码都写在 struts.xml 文件中。因此在实际开发中，每个开发组成员都可以创建配置文件，然后将多个配置文件包含到 struts.xml 文件中，完成整个应用的配置。

include 标签的格式如下：

```
<include file="配置文件路径"/>
```

1．配置 main.xml

根据图 19.2 所示，在 src 路径上创建 main.xml 文件，配置 Action，代码如下：

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
    "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
    <constant name="struts.devMode" value="true" />
    <package name="admin" extends="struts-default" namespace="/admin">
        <!--管理员登录系统-->
        <action name="index" class="com.cvit.BBS.action.Admin_Action"
            method="login">
            <!--管理员登录系统成功-->
            <result>/main/index.jsp</result>
            <!--管理员登录系统失败-->
            <result name="input">/main/login.jsp</result>
        </action>
        <!--用户管理模块主页-->
        <action name="User_manage">
            <result>/main/User_manage.jsp</result>
        </action>
        <!--添加用户输入页-->
        <action name="User_add_input">
            <result>/main/User_add_input.jsp</result>
        </action>
        <!--在修改用户信息前,查找用户信息-->
        <action name="User_query" class="com.cvit.BBS.action.User_Action"
            method="queryById">
            <result>/main/User_modify_input.jsp</result>
        </action>
        <!--使用通配符简化“增、删、改、列表”的配置-->
        <action name=" * _ * " class="com.cvit.BBS.action.{1}_Action"
            method="{2}">
            <result>/main/{1}_{2}_success.jsp</result>
        </action>
    </package>
</struts>
```

2．配置 struts.xml

在 struts.xml 文件中，配置主要代码如下：

```
<struts>
```

```
    <constant name="struts.devMode" value="true" />
    <include file="main.xml"/>
</struts>
```

include 标签的 file 属性,指定被包含的配置文件路径;main.xml 和 struts.xml 在同一目录下,因此其相对路径为 main.xml。

19.3.5 编写视图层文件的请求

根据 struts.xml 文件中对 Action 的配置,编写视图层文件中的请求代码。

1. login.jsp

主要代码如下:

```
<body>
    <form action="admin/index" method="post">
        ...
    </form>
</body>
```

2. index.jsp

index.jsp 为后台主页,主要代码如下:

```
<body>
    欢迎 *** 登录本系统<br/>
    <a href="admin/User_manage">会员管理</a><br/>
    ...
</body>
```

3. user_manage.jsp

user_manage.jsp 为用户管理模块主页,主要代码如下:

```
<body>
    <a href="admin/user_add_input">添加会员</a><br />
    <a href="admin/user_list">显示会员列表</a><br />
</body>
```

4. user_add_input.jsp

user_add_input.jsp 为添加用户页,主要代码如下所示:

```
<body>
    <form action="admin/user_add" method="post">
        ...
    </form>
</body>
```

5. user_list_success.jsp

user_list_success.jsp 为显示用户列表页,主要代码如下:

```
<body>
    会员列表<br/>
    会员 1 信息<a href="admin/user_query">修改会员</a>|
<a href="admin/User_delete"> 删除会员</a><br/>
</body>
```

19.3.6　测试架构

部署工程，在浏览器地址栏中输入 http://localhost:8080/BBS_Struts2_2000_Framework/main/login.jsp，打开 login.jsp 页面（见图 19.3）。单击“登录”按钮，跳转到 index.jsp 页面（见图 19.4）。单击“会员管理”超链接，打开 user_manage.jsp（见图 19.5）。单击“添加会员”超链接，打开 user_add_input.jsp（见图 19.6）。在 user_manage.jsp 页单击“显示会员列表”超链接，打开 user_list_success.jsp（见图 19.7）。

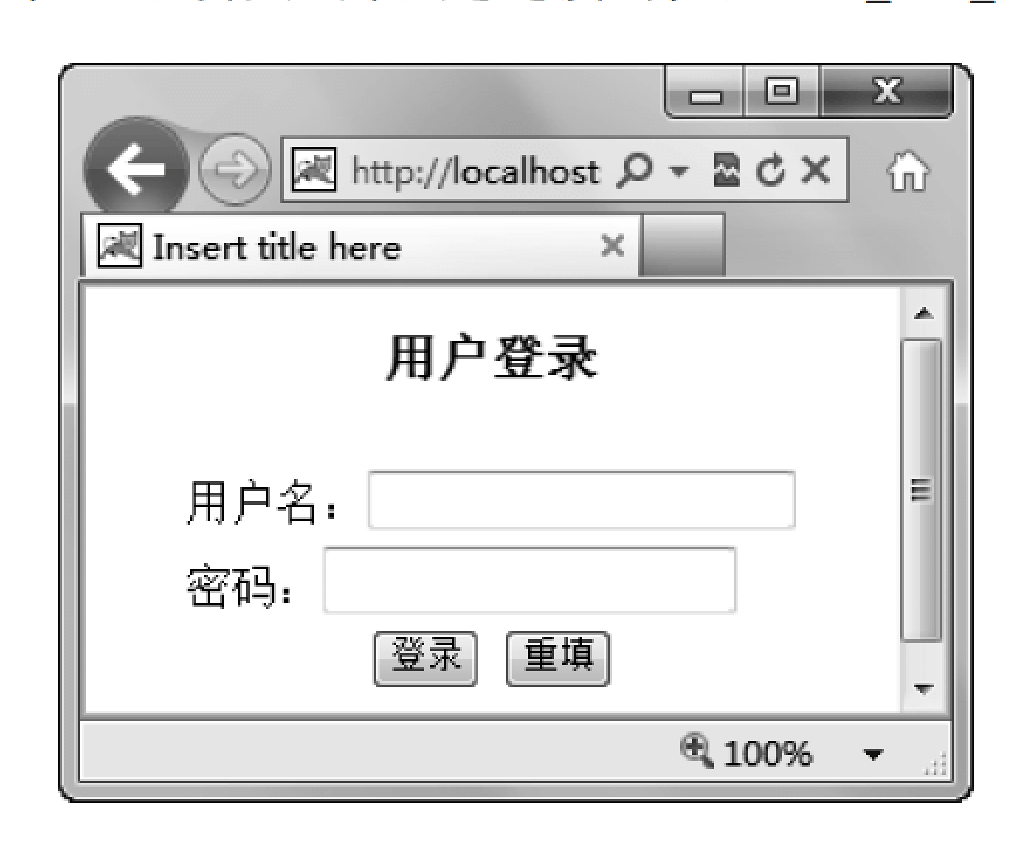


图 19.3　login.jsp 页面

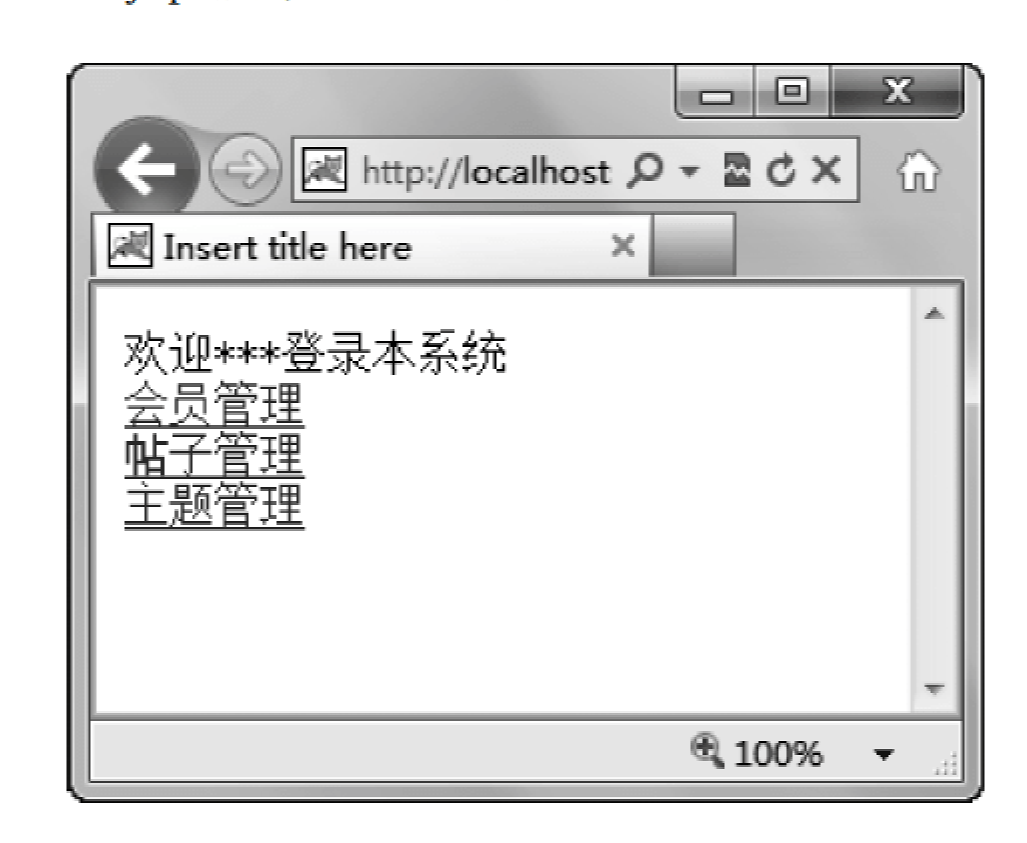


图 19.4　index.jsp 页面

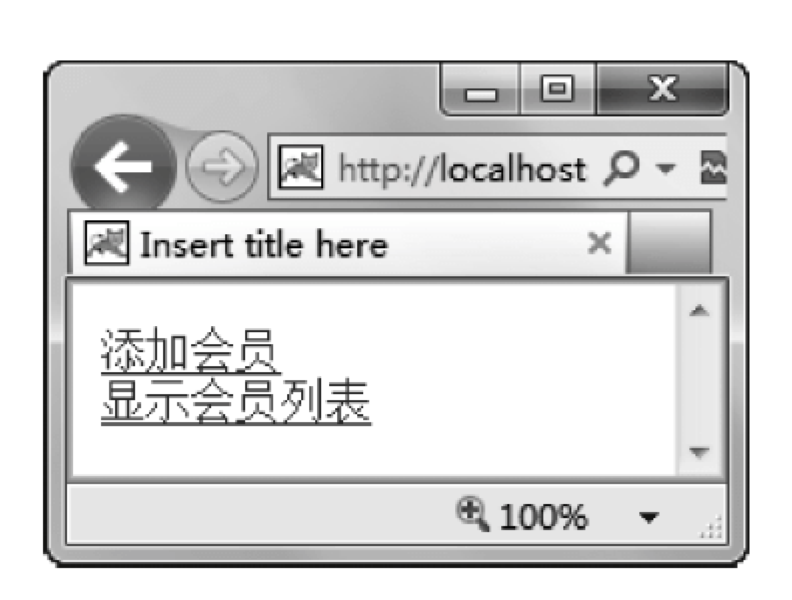


图 19.5　user_manage.jsp 页面



图 19.6　user_add_input.jsp 页面

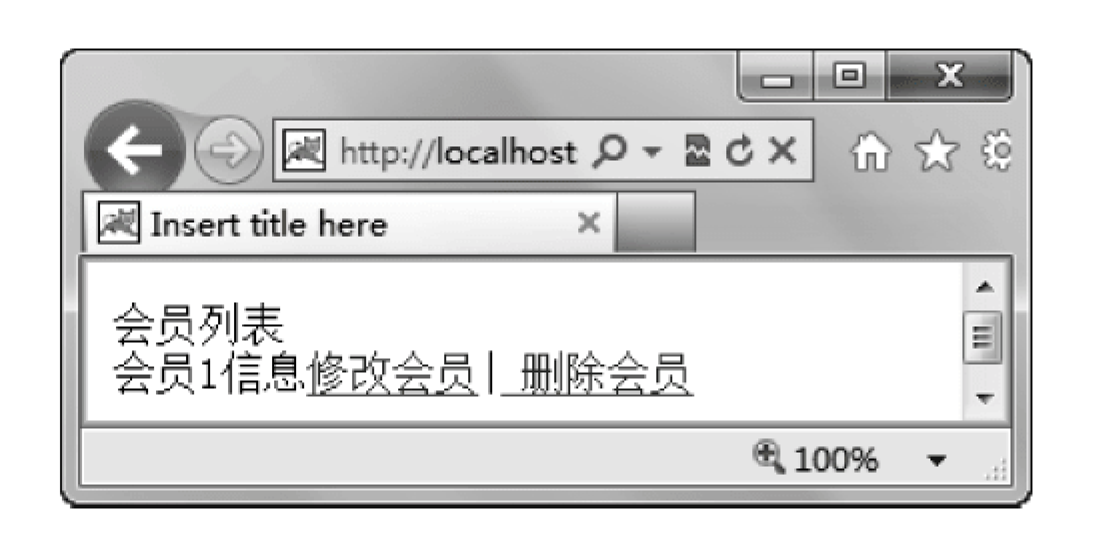


图 19.7 user_list_success.jsp 页面

19.3.7 拓展任务 1：完成用户管理模块架构的实现

1. 任务要求

将完成本节未完成的删除用户、修改用户和退出系统架构实现。

2. 建议步骤

（1）使用工程 BBS_Struts2_2000_Framework。

（2）创建视图层文件，并根据 struts.xml 文件中 Action 配置，编写请求。

（3）部署工程。

（4）调试系统。打开浏览器，输入客户端请求，测试程序，如有错误，进行修改。

19.3.8 实训 1：实现论坛管理系统的主题管理模块的架构

实训要求：

（1）设计论坛管理系统的主题管理模块的架构。

（2）创建视图层文件。

（3）创建 Action 类。

（4）配置 Action。

（5）编写视图层文件的请求代码。

（6）测试。

19.4 Struts 2 框架的 MVC 各层的实现

系统架构搭建完成后，完成 MVC 各层的实现，包括以下几步。

（1）创建模型类。

（2）实现业务逻辑组件。创建工具类和业务逻辑类，实现业务逻辑的处理。在完成创建模型层和业务逻辑后，可以测试业务逻辑是否正确实现。

提示：在开发系统的过程中，应尽量做到每做一点改动，就及时测试，以保证每一步都正确完成，以防止在大量代码中查找错误，降低调试程序的效率。

（3）编写 Action，接收参数，调用业务逻辑类的相应方法，并设置向页面传递的相关

数据。

(4) 编辑视图层,实现数据输出、输入等功能。

19.4.1　创建模型类

根据数据字典中会员表的设计,定义 User 类,代码如下:

```
package com.cvit.BBS.model;
public class User {
    private int id;
    private String name;
    private String password;
    private int age;
    //属性的 getter 和 setter 方法略
    …
}
```

19.4.2　实现业务逻辑

1. 创建 DB 类

创建数据库访问工具类 DB,代码如下:

```
package com.cvit.BBS.util;
import java.sql.*;
public class DB {
    //连接数据库
    public static Connection getConn() {
        Connection conn = null;
        try {
            Class.forName("com.mysql.jdbc.Driver");
            conn = DriverManager
        .getConnection("jdbc:mysql://localhost/DB_BBS?user=root&password=root");
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return conn;
    }

    //创建 Statement 对象
    public static Statement getStatement(Connection conn) {
        Statement stmt = null;
        try {
            if (conn != null) {
                stmt = conn.createStatement();
            }
```

```
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return stmt;
    }

    //查询记录
    public static ResultSet getResultSet(Statement stmt, String sql) {
        ResultSet rs = null;
        try {
            if (stmt != null) {
                rs = stmt.executeQuery(sql);
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return rs;
    }

    //删除、修改记录
    public static void executeUpdate(Statement stmt, String sql) {
        try {
            if (stmt != null) {
                stmt.executeUpdate(sql);
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    //执行带参数的 SQL 语句
    public static PreparedStatement prepare(Connection conn, String sql) {
        PreparedStatement pstmt = null;
        try {
            if (conn != null) {
                pstmt = conn.prepareStatement(sql);
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return pstmt;
    }

    //关闭连接
    public static void close(Connection conn) {
        try {
            if (conn != null) {
                conn.close();
                conn = null;
```

```
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    //关闭连接
    public static void close(Statement stmt) {
        try {
            if (stmt != null) {
                stmt.close();
                stmt = null;
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    //关闭连接
    public static void close(ResultSet rs) {
        try {
            if (rs != null) {
                rs.close();
                rs = null;
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

2. 创建 UserService 类

根据业务需求，创建 UserService 类，实现对会员表的增、删、改、查、列表等功能。实现添加用户和显示用户列表功能的代码如下：

```
package com.cvit.BBS.service;
import java.sql.*;
import java.util.ArrayList;
import java.util.List;
import com.cvit.BBS.model.User;
import com.cvit.BBS.util.DB;
public class UserService {
    public void add(User user) {
        System.out.println("------UserService.add()------");
        System.out.println(user.getName());

        Connection conn=DB.getConn();
        String sql="insert into _user values(null,?,?,?)";
        PreparedStatement ps=DB.prepare(conn, sql);
```

```
        try {
            ps.setString(1,user.getName());
            ps.setString(2, user.getPassword());
            ps.setInt(3,user.getAge());
            ps.executeUpdate();
        } catch (SQLException e) {
            //TODO Auto-generated catch block
            e.printStackTrace();
        }
        DB.close(ps);
        DB.close(conn);
    }

    public List<User>  list() {
        Connection conn = DB.getConn();        //连接数据库
        Statement stmt=DB.getStatement(conn);
        String sql = "select * from _user";
            List<User> users = new ArrayList<User>();   //定义一个对象数组集合
            ResultSet rs =DB. getResultSet(stmt,sql);//从数据库中取出所有记录,放入 rs 中
            User u = null;                      //定义一个对象,将 rs 中取出的记录放入 u 中
            try {
                while(rs.next()) {
                    u = new User();
                    u.setId(rs.getInt(1));
                    u.setName(rs.getString(2));
                    u.setPassword(rs.getString(3));
                    u.setAge(rs.getInt(4));
                    users.add(u);               //将 u 对象中的记录放入集合
                }
            } catch (SQLException e) {
                //TODO Auto-generated catch block
                e.printStackTrace();
            }
            DB.close(rs);
            DB.close(conn);
        return users;                           //返回的类型为 users
    }
    public void delete(int id) {
    }
    public void modify(User user) {
    }

    public User queryById(int id) {
        return null;
    }
}
```

在创建 UserService 类后，可以如 delete()方法一样，先创建方法，而不写功能语句，以方便测试 Action 调用方法、参数传递等是否正确。

19.4.3　实现 Action 功能

1. 创建 UserDTO 类

Action 类从页面接收参数传递，在创建用户时，因为 User 类不能满足参数传递需求，需要创建 UserDTO 类传递数据，主要代码如下：

```
package com.cvit.BBS.VO;
public class UserDTO {
    private int id;
    private String name;                    //会员名
    private String password;                //密码
    private String password2;               //密码确认
    private int age;                        //年龄
    //属性的 getter 和 setter 方法略
    ...
}
```

如果模型类可以满足参数传递的需求，这一步可以省略。

2. 修改 User_Action

修改 User_Action，接收参数，并调用 UserService 类的相应方法实现功能。

以添加用户为例，创建 UserDTO 类的对象，用于接收参数，调用 UserService 类的 add()方法实现将用户信息插入数据库中，调用 UserService 类的 list()方法实现从数据库中读取所有用户信息，主要代码如下：

```
public class User_Action extends ActionSupport {
    private UserService userService = new UserService();
    private User user;
    private UserDTO userDTO;
    private List<User> users;
    public String add() {
        user = new User();
        user.setName(userDTO.getName());
        user.setPassword(userDTO.getPassword());
        user.setAge(userDTO.getAge());
        userService.add(user);
        return SUCCESS;
    }
    public String list() {
        users=userService.list();
        return SUCCESS;
    }
    ...
}
```

3. 修改 Admin_Action 类

修改 Admin_Action 类,实现管理员登录功能,主要代码如下:

```
public class Admin_Action extends ActionSupport {
    private String username;
    private String password;
    public String login() {
        if(username.equals("aaa")&&password.equals("123")){
            ActionContext ctx = ActionContext.getContext();
            Map session = ctx.getSession();
            session.put("user",username);
            return SUCCESS;
        }
        else {
            this.addFieldError("loginFail", "您输入的用户名或密码不正确,请重新输入");
            return INPUT;
        }
    }
    //属性 getter 和 setter 方法略
}
```

在实际开发中,Admin_Action 类的 login()方法应调用业务逻辑类的相应方法,到数据库中查询管理员是否存在,此处鉴于篇幅所限,将业务逻辑在 Action 类中模拟实现。

另外,此步骤可以再分成两步完成:首先定义数据传递类 DTO,测试数据传递是否正确;然后再调用业务逻辑,测试数据返回是否正确。

19.4.4 实现视图层功能

在完成以上步骤后,最后一步编写视图层 JSP 文件,实现提交参数和输出数据功能。下面以管理员登录、添加用户和显示用户列表为例讲解。

1. 管理员登录

打开 login.jsp,修改代码如下:

```
<body>
    <form action="admin/index" method="post">
    <s:property value="errors.loginFail[0]"/>
    ...
        <tr><td>用户名:<input type="text" name="username"/></td></tr>
        <tr><td>密码:<input type="password" name="password"/></td></tr>
    ...
    </form>
</body>
```

使用<property>标签输出登录失败提示信息,使用<input>标签的 name 属性传递参数。

2. 添加用户

打开 user_add_input.jsp,使用<input>标签的 name 属性传递参数,修改代码如下:

```
<body>
    ...
        <tr><td>用户名：<input type="text" name="userDTO.name"/></td></tr>
        <tr><td>密码：<input type="password" name="userDTO.password"/></td>
          </tr>
        <tr><td>确认密码：<input type="password" name="userDTO.password2"/></td>
        </tr><tr><td>年龄：<input type="text" name="userDTO.age"/></td>
    ...
  </form>
</body>
```

3. 显示用户信息列表

打开 user_add_input.jsp，修改代码如下：

```
<body>
    会员列表  <br />
    <s:if test="users.size==0">无记录</s:if>
    <s:else>
        <s:iterator value="users" var="user">
            会员名：<s:property value="#user.name" /> |
            会员年龄：<s:property  value="#user.age" />|
            <a href="admin/User_query">修改会员</a>|
            <a href="admin/User_delete">删除会员</a><br />
        </s:iterator>
    </s:else>
</body>
```

使用 Struts 2 标签 if、else、iterator 和 property 显示用户列表。

19.4.5 测试

部署工程，在浏览器地址栏中输入 http://localhost:8080/BBS_Struts2_2000_Framework/main/login.jsp，打开 login.jsp 页面(见图 19.3)。输入错误的用户名和密码，单击“登录”按钮，返回 login.jsp 页，并显示登录失败提示信息(见图 19.8)。输入正确的用户名“aaa”和密码“123”，单击“登录”按钮，跳转到 index.jsp 页面(见图 19.4)。

在 index.jsp 页面单击“会员管理”超链接，打开 user_manage.jsp，如图 19.5 所示。

在 user_manage.jsp 页面中单击“添加会员”超链接，打开 user_add_input.jsp，如图 19.6 所示。

在 user_add_input.jsp 页面中输入用户名、密码、确认密码和年龄，单击“添加”按钮，将用户信息添加到数据库中。

在 user_manage.jsp 页面中单击“显示会员列表”超链接，打开 user_list_success.jsp，显示会员列表，当无会员时，显示“无记录”(见图 19.9)。当使用 user_add_input.jsp 添加会员后，数据库中有会员信息，显示会员名和年龄，如图 19.10 所示。



图 19.8 管理员登录失败

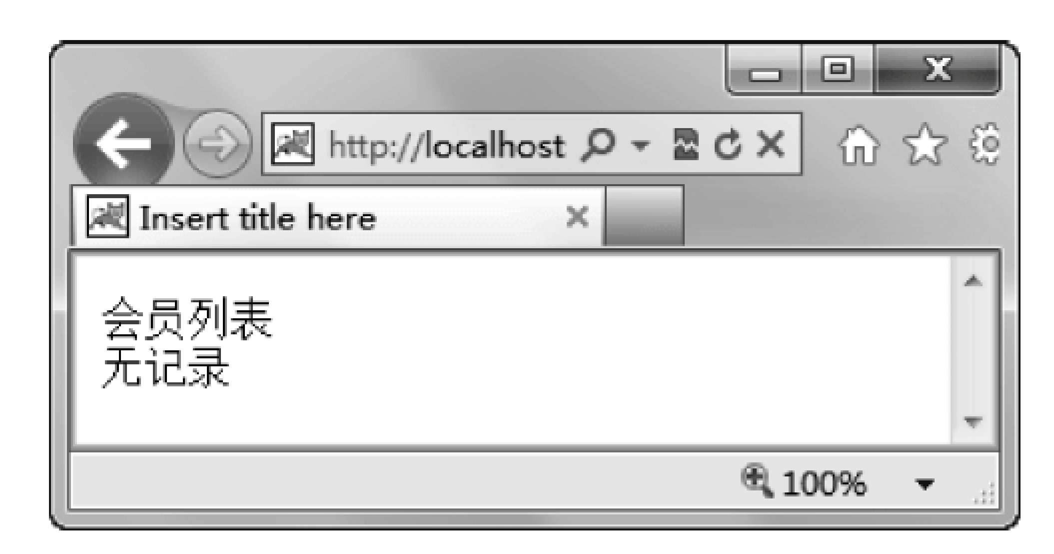


图 19.9 无会员时 user_list_success.jsp 页面显示的信息

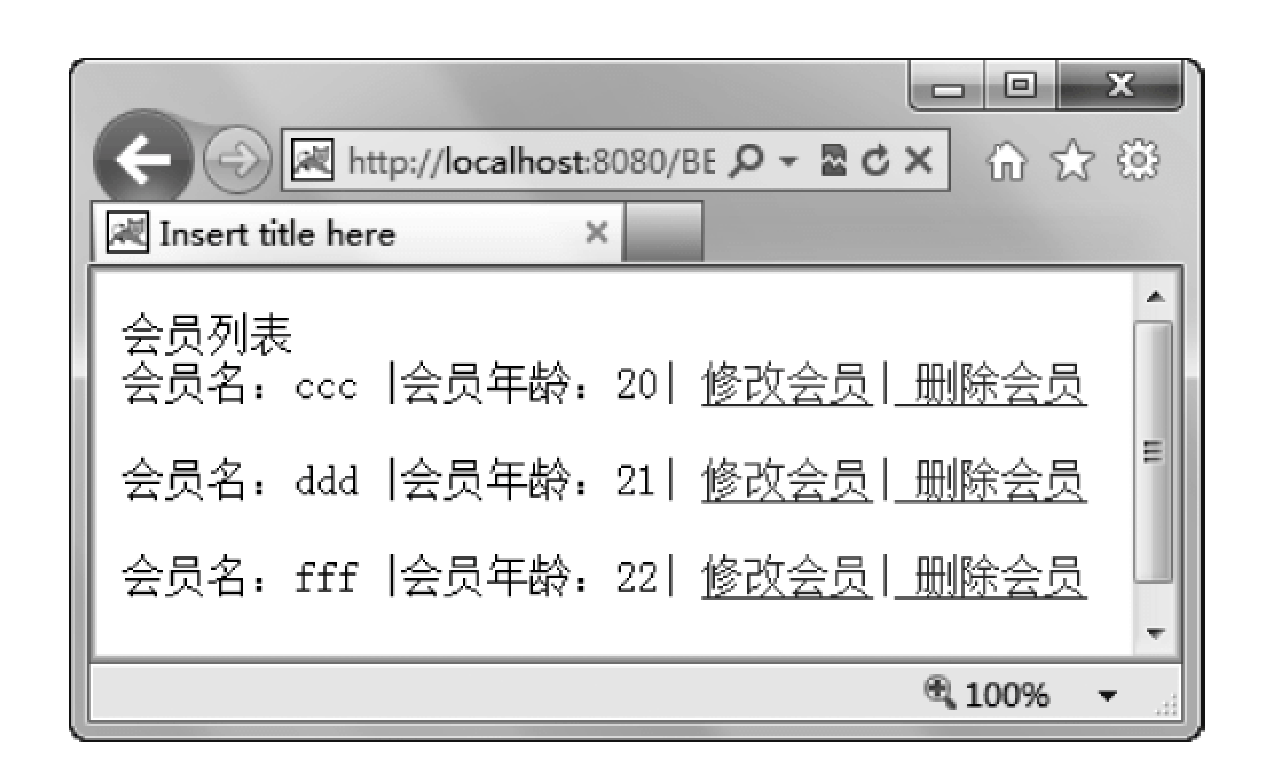


图 19.10 有会员时 user_list_success.jsp 页面显示的信息

19.4.6 拓展任务 2：完成用户管理模块 MVC 各层的实现

1. 任务要求

完成删除用户、修改用户和退出系统的功能。

2. 建议步骤

(1) 使用工程 BBS_Struts2_2000_Framework。

(2) 编写 UserService，完成 delete()方法、modify ()方法和 queryById()方法。

(3) 编写 Action 类，定义属性用于传递数据，完成 delete()方法、modify()方法、queryById()和 quit()方法。

(4) 编写视图层文件，完成请求参数的设置和数据显示。

(5) 部署工程。

(6) 调试系统。打开浏览器，输入客户端请求，测试程序，如有错误，进行修改。

19.4.7 根据需求变动修改代码并测试

需求的变更在系统开发过程中是不可避免的，例如当删除、修改、添加用户成功后，不

显示成功页面，而是返回 user_list_success.jsp 页面，并显示删除、修改或添加用户成功后新的用户列表信息。

1. 修改 Action 配置

使用 Struts 框架实现这个变更，只需要修改配置文件 struts.xml 中 Action 的配置。例如，当添加用户成功后，配置 Action 的结果类型为 chain 或 redirectAction，将请求交给 User_list Action 处理，在 main.xml 文件中添加以下配置代码。

```
<package name="admin" extends="struts-default" namespace="/admin">
    ...
    <action name="User_add" class="com.cvit.BBS.action.User_Action"
        method="add">
        <result type="chain">User_list</result>
    </action>
    ...
</package>
```

当客户端提交添加用户的请求时，调用 User_Action 的 add()方法，将新增的用户信息插入数据库中，再调用 User_Action 的 list()方法，打开 user_list_success.jsp 文件，显示添加用户成功后新的用户信息列表。

2. 测试

打开 user_add_input.jsp 页面，添加用户信息，如图 19.11 所示。

单击“添加”按钮，跳转到 user_list_success.jsp，显示添加后的用户信息列表，如图 19.12 所示。



图 19.11　user_add_input.jsp 页面

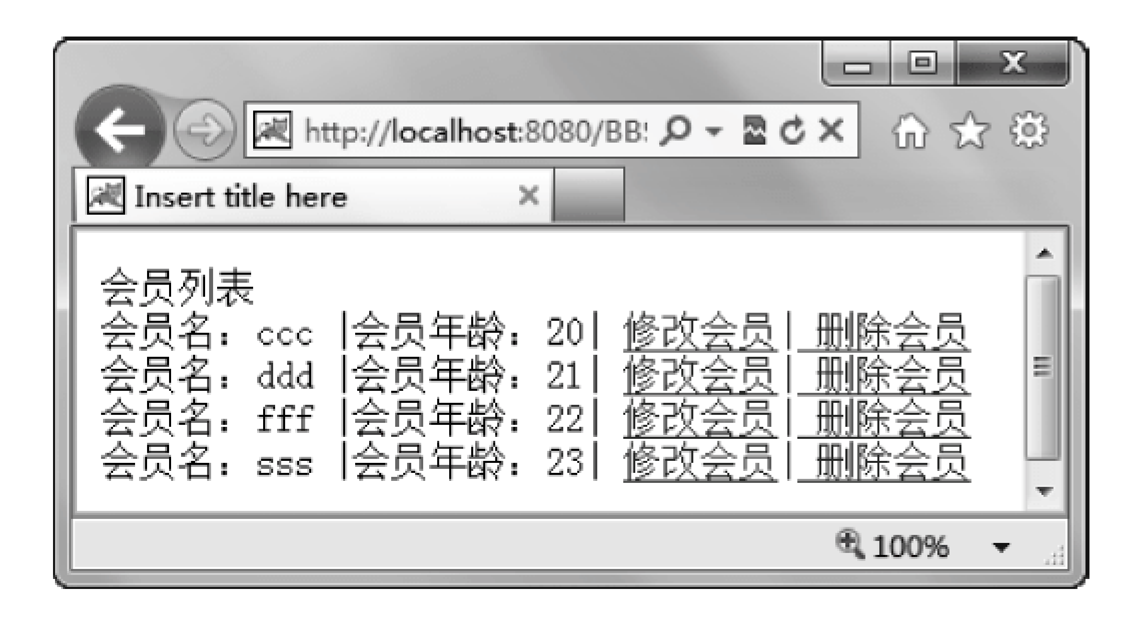


图 19.12　user_list_success.jsp 页面

19.4.8　实训 2：实现论坛管理系统的主题管理模块 MVC 各层

实训步骤：

（1）实现模型层，定义 Theme 类。

（2）实现业务逻辑，定义 ThemeService 类。

(3) 实现 Action 功能，定义 Theme_Action 类。
(4) 实现视图层功能。
(5) 测试。

19.5 实现 Struts 2 框架的数据输入校验

19.5.1 使用 validateXxx()方式实现输入校验

当管理员登录系统时，输入的用户名和密码不能为空。下面采用 validateXxx()的方式实现，实现步骤如下。

1. 在 Action 类中，插入输入校验方法

在 Admin_Action 类中，插入方法 validateLogin()，代码如下：

```
public void validateLogin() {
    if(username==null||username.equals("")){
        this.addFieldError("username", "用户名不能为空");
    }
    if(password==null||password.equals("")){
        this.addFieldError("password", "密码不能为空");
    }
}
```

2. 配置 input 结果

当数据输入校验失败时，返回逻辑视图 input，因为<result name="input">/main/login.jsp</result>已经配置，所以配置文件代码不必修改，代码如下：

```
<action name="index" class="com.cvit.BBS.action.Admin_Action" method="login">
    <result>/main/index.jsp</result>
    <result name="input">/main/login.jsp</result>
</action>
```

3. 设置页面显示提示信息

设置页面显示提示信息的方式有两种：一种是使用 FieldError 或 property 标签输出提示信息；另一种是使用 Struts 表单标签，则输出信息自动显示。

(1) 使用 FieldError 或 property 标签输出提示信息。打开 login.jsp，插入 FieldError 或 property 标签，代码如下：

```
<body>
<form action="admin/index" method="post">
<s:property value="errors.loginFail[0]"/>
<table align="center">
    <caption><h3>用户登录</h3></caption>
    <tr>
```

```
        <td>用户名：<input type="text" name="username"/>
        <s:fielderror fieldName="username"/></td>
    </tr>
    <tr>
        <td>密码：<input type="password" name="password" />
        <s:property value="fieldErrors.password" /></td>
    </tr>
    <tr align="center">
        <td><input type="submit" value="登录"/>
            <input type="reset" value="重填" /></td>
    </tr>
</table>
</form>
</body>
```

部署工程，在浏览器地址栏中输入 http://localhost:8080/BBS_Struts2_2000_Framework/main/login.jsp，打开 login.jsp 页面。不输入用户名和密码，单击“登录”按钮，返回 login.jsp 页面，并显示提示信息，如图 19.13 所示。



图 19.13　login.jsp 显示数据输入校验提示信息

（2）使用 Struts 2 表单标签，代码如下：

```
<s:form action="admin/index" method="post">
    <s:property value="errors.loginFail[0]"/>
    <caption><h3>用户登录</h3></caption>
    <s:textfield name="username" label="用户名"/><br/>
    <s:password name="password" label="密码"/><br/>
    <s:submit value="登录"/><br/><s:reset  value="重填" />
</s:form>
```

使用 Struts 2 表单标签，不必插入 FieldError 或 property 标签，textfield 和 password 标签自动显示提示信息，如图 19.14 所示。



图 19.14　使用 Struts 2 表单标签显示数据输入校验提示信息

19.5.2　使用校验框架实现输入校验

在 Struts 2 应用中，更推荐使用校验框架实现输入校验，以减少程序代码。

下面以添加用户的输入校验为例，练习使用校验框架实现输入校验。

由于 UserAction 在传递数据时，使用 UserDTO 类的对象传递数据，因此使用 Visitor 校验器进行数据输入校验。

1. 创建并编写 User_Action-User_add-validation.xml

在与 User_Action 同路径下，创建 User_Action-User_add-validation.xml。注意文件名中“User_Action”与 User_Action 类名一致，“User_add”与配置文件 struts.xml 中调用 UserAction 的 add()方法的 Action 配置名相同。

UserAction-User_add-validation.xml 配置文件代码如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE validators PUBLIC "-//OpenSymphony Group//XWork Validator 1.0.2//EN"
"http://www.opensymphony.com/xwork/xwork-validator-1.0.2.dtd">
<validators>
<!-- 指定校验 userDTO 字段-->
<field name="userDTO">
<!--使用 visitor 校验器-->
    <field-validator type="visitor">
        <!-- 指定校验规则文件-->
        <param name="context">userContext</param>
        <!-- 指定校验失败后提示信息是否添加下面前缀-->
        <param name="appendPrefix">true</param>
        <message>您输入的用户信息中: </message>
    </field-validator>
</field>
</validators>
```

2. 创建并编写 UserDTO-userContext-validation. xml

在与 UserDTO. java 同路径下，创建 UserDTO-userContext-validation. xml 配置文件，其中"userContext"与<param name="context">userContext</param>一致。

User-userContext-validation. xml 配置代码如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE validators PUBLIC "-//OpenSymphony Group//XWork Validator 1.0.2//EN"
"http://www.opensymphony.com/xwork/xwork-validator-1.0.2.dtd">
<validators>
    <!-- 校验 UserDTO 的 name 属性 -->
    <field name="name">
        <!-- 指定 name 属性必须满足必填规则 -->
        <field-validator type="requiredstring">
            <param name="trim">true</param>
            <!--指定校验失败后提示信息-->
            <message>用户名必须输入</message>
        </field-validator>
        <!-- 指定 name 属性必须匹配正则表达式 -->
        <field-validator type="regex">
            <param name="expression"><![CDATA[(\w{3,8})]]></param>
            <!-- 指定校验失败后提示信息-->
            <message>
            用户名只能包含字母和数字，且长度必须在 3 到 8 之间
            </message>
        </field-validator>
    </field>
    <!-- 校验 UserDTO 的 password 属性 -->
    <field name="password">
        <!-- 指定 password 属性必须满足必填规则 -->
        <field-validator type="requiredstring">
            <param name="trim">true</param>
            <!--指定校验失败后提示信息-->
            <message>密码必须输入</message>
        </field-validator>
        <!-- 指定 password 属性必须满足匹配指定的正则表达式 -->
        <field-validator type="regex">
            <param name="expression"><![CDATA[(\w{6,10})]]></param>
            <!-- 指定校验失败后提示信息-->
            <message>
            密码只能包含字母和数字，且长度必须在 6 到 10 之间
            </message>
        </field-validator>
    </field>
    <!-- 校验 UserDTO 的 password2 属性 -->
    <field name="password2">
        <!-- 指定 password2 属性必须满足必填规则 -->
        <field-validator type="requiredstring">
            <param name="trim">true</param>
```

```
            <!--指定校验失败后提示信息-->
            <message>密码必须输入</message>
        </field-validator>
        <!-- 指定 password2 属性必须满足匹配指定的正则表达式 -->
        <field-validator type="regex">
            <param name="expression"><![CDATA[(\w{6,10})]]></param>
            <!-- 指定校验失败后提示信息-->
            <message>
            确认密码只能包含字母和数字,且长度必须在 6 到 10 之间
            </message>
        </field-validator>
        <!-- 校验 UserDTO 的 password 和 password2 属性值是否一致-->
        <field-validator type="fieldexpression">
            <param name="expression">
                <![CDATA[(password2.equals(password)) ]]>
            </param>
            <!--指定校验失败后提示信息-->
            <message>密码和确认密码必须一致</message>
        </field-validator>
    </field>
    <!-- 指定 UserDTO 的 age 属性必须在指定范围内-->
    <field name="age">
        <field-validator type="int">
            <param name="min">0</param>
            <param name="max">100</param>
            <!-- 指定校验失败后提示信息-->
            <message>年龄必须在 0 到 100 岁之间</message>
        </field-validator>
    </field>
</validators>
```

3. 配置 result

数据输入校验失败后,返回 input,因此在 main. xml 中,配置 result 结果。当数据输入校验失败后,返回 user_add_input. jsp,代码如下:

```
<action name="User_add" class="com.cvit.BBS.action.User_Action"
    method="add">
    <result name="input">/main/user_add_input.jsp</result>
    <result type="chain">User_list</result>
</action>
```

4. 在视图层文件中编写显示提示信息代码

打开 User_add_input. jsp,插入 fielderror 标签,显示提示信息,代码如下:

```
<body>
   <form action="user/User_add" method="get">
       用户名: <input type="text" name="user.name"/>
       <s:fielderror fieldName="user.name" theme="simple"/><br>
       密  码: <input type="password" name="user.password"/>
```

```
        <s:fielderror fieldName="user.password" theme="simple"/><br>
        年　龄：<input type="text" name="user.age"/>
        <s:fielderror fieldName="user.age" theme="simple"/><br>
        <input type="submit" value="添加">
    </form>
</body>
```

当不输入用户名、密码、确认密码，年龄输入不在 0～100 之间时，显示提示信息，如图 19.15 所示。当输入的密码和确认密码不符合长度要求，且密码和确认密码不一致时，显示提示信息，如图 19.16 所示。

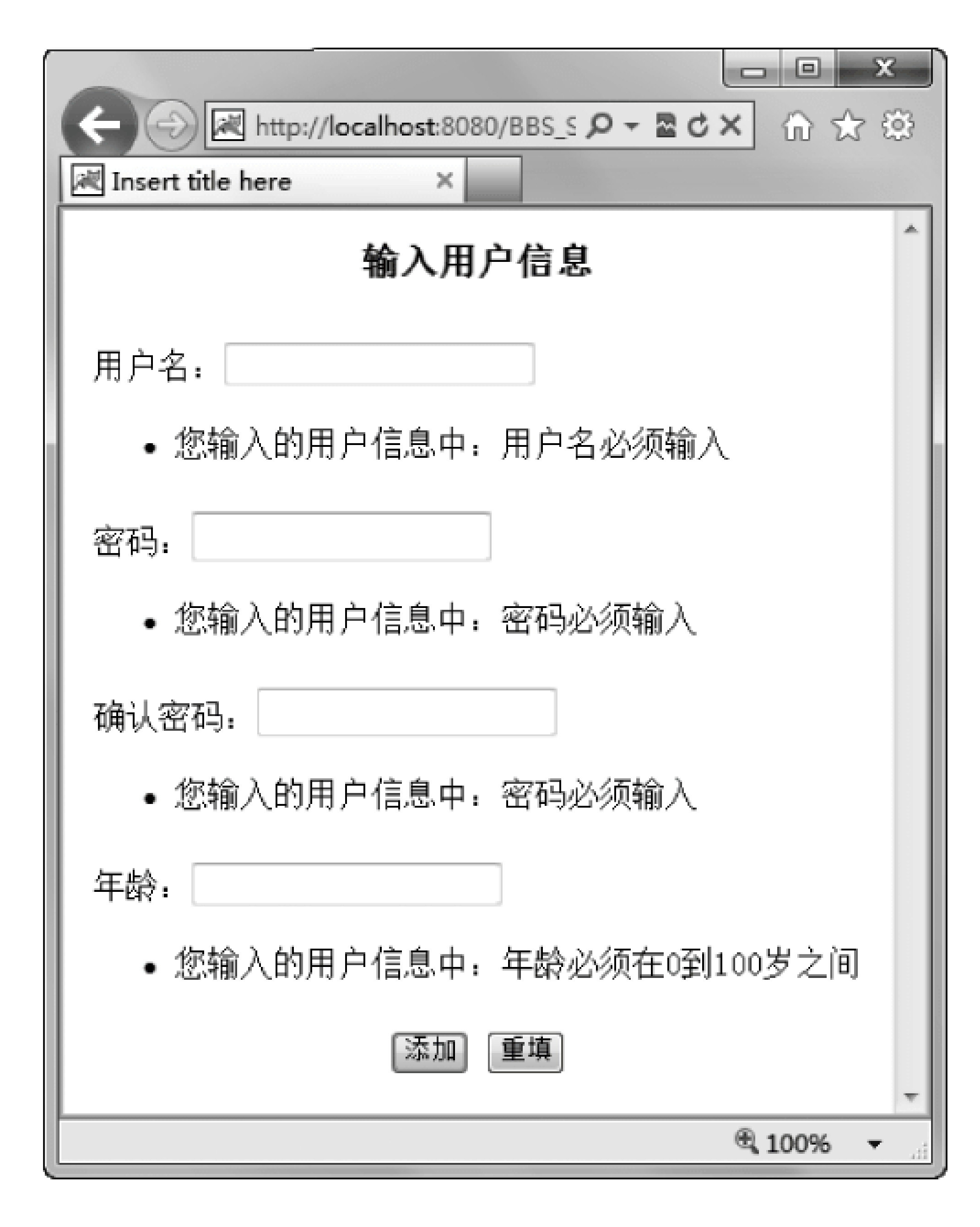


图 19.15　年龄不在 0～100 之间的提示信息

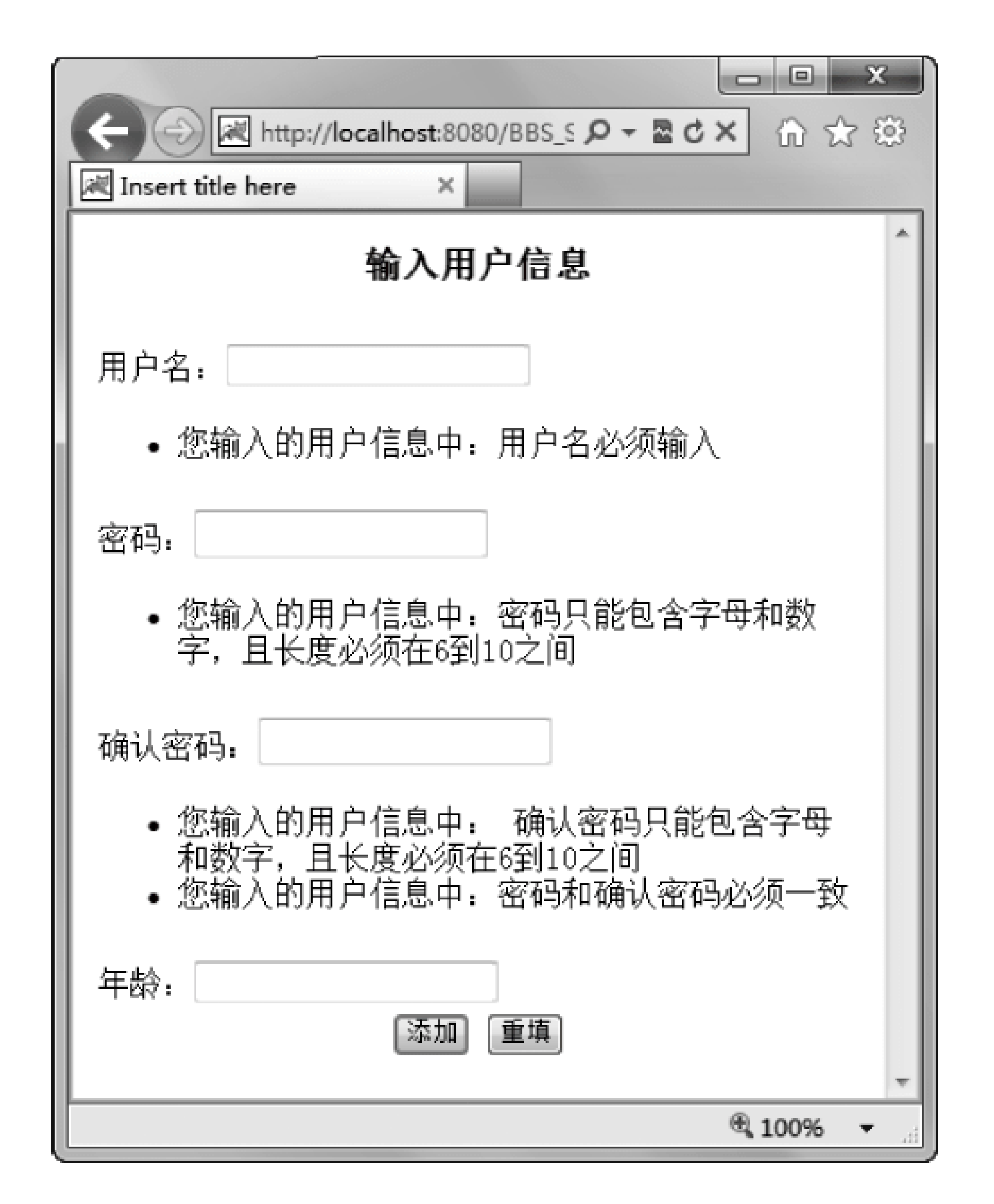


图 19.16　两次输入的密码不一致的提示信息

19.5.3　拓展任务 3：完成用户管理模块数据输入校验

1. 任务要求

完成修改用户的数据输入校验。

2. 建议步骤

(1) 使用工程 BBS_Struts2_2000_Framework。

(2) 创建并编写 User_Action-User_modify-validation.xml。

(3) User-userContext-validation.xml 已经编写完成，不需要修改。

(4) 配置 result。

(5) 编写视图层文件，显示数据输入校验失败的提示信息。

(6) 部署工程。

（7）调试系统。打开浏览器，输入客户端请求，测试程序，如有错误，进行修改。

19.5.4 实训 3：实现论坛管理系统的主题管理模块的数据输入校验

实训步骤：

（1）使用工程 BBS_Struts2_2000_Framework。

（2）创建并编写 Theme_Action-Theme_add-validation.xml 和 Theme_Action-Theme_modify-validation.xml。

（3）编写 Theme-themeContext-validation.xml，配置 Struts 2 内置校验器。

（4）配置 result。

（5）编写视图层文件，显示数据输入校验失败的提示信息。

（6）测试。

19.6 配置 Struts 2 框架的拦截器

19.6.1 创建与应用拦截器

下面配置论坛管理系统的权限验证拦截器。

1. 创建拦截器类

在 com.cvit.struts2.interceptor 包下创建拦截器类 AuthorityInterceptor，代码如下：

```
package com.cvit.struts2.interceptor;
import java.util.Map;
import com.opensymphony.xwork2.ActionInvocation;
import com.opensymphony.xwork2.interceptor.AbstractInterceptor;
public class AuthorityInterceptor extends AbstractInterceptor{
    @Override
    public String intercept(ActionInvocation invocation) throws Exception {
        Map session=invocation.getInvocationContext().getSession();      //获得 session
        String user=(String) session.get("user");    //从 session 中获得 user
        if(user!=null){//如果 user 不为空，则用户已经登录，继续执行
            return invocation.invoke();
        }
        //否则将提示信息放到 session 中，并返回“login”逻辑视图，不执行 Action
        session.put("tip", "您还没有登录系统，请输入用户名和密码登录");
        return "login";
    }
}
```

2. 部署拦截器

打开配置文件 struts.xml，部署拦截器 AuthorityInterceptor，并配置拦截器栈。另

外，因为配置全局结果 login，代码如下：

```
<package name="myDefault" extends="struts-default">
    <interceptors>
        <interceptor name="AuthorityInterceptor"
                class="com.cvit.struts2.interceptor.AuthorityInterceptor"/>
        <interceptor-stack name="mydefault">
            <interceptor-ref name="defaultStack" />
            <interceptor-ref name="AuthorityInterceptor"/>
        </interceptor-stack>
    </interceptors>
        <global-results>
            <!-- 当返回 login 逻辑视图时，转入 login.jsp 页面 -->
            <result name="login">/main/login.jsp</result>
        </global-results>
</package>
```

3. 配置 Action，应用拦截器

打开配置文件 struts.xml，输入以下代码。

```
<package name="main" extends="myDefault" namespace="/">
    <action name="login" class="com.cvit.struts2.action.UserAction" method="login">
        <result>/index.jsp</result>
        <result name="fail">/login.jsp</result>
    </action>

    <action name="list" class="com.cvit.struts2.action.ThemeAction" method="list">
        <result>/Theme_list_success.jsp</result>
        <interceptor-ref name="mydefault"/>
    </action>
    <!-- 退出系统 -->
    <action name="quit" class="com.cvit.struts2.action.UserAction" method="quit">
        <result>/login.jsp</result>
        <interceptor-ref name="mydefault"/>
    </action>
</package>
```

4. 编写视图层文件，显示提示信息

打开 login.jsp，插入 property 标签，代码如下：

```
<body>
    <form action="admin/index" method="post">
        <s:property value="errors.loginFail[0]"/>
        <s:property value="#session.tip"/>
        <table align="center">
            <caption><h3>用户登录</h3></caption>
    ...
</body>
```

5. 测试

部署工程,在浏览器的地址栏中输入 http://localhost:8080/BBS_Struts2_2000_Framework/main/user_add_input,跳转到 login.jsp 页面,并显示请登录系统提示信息,如图 19.17 所示。

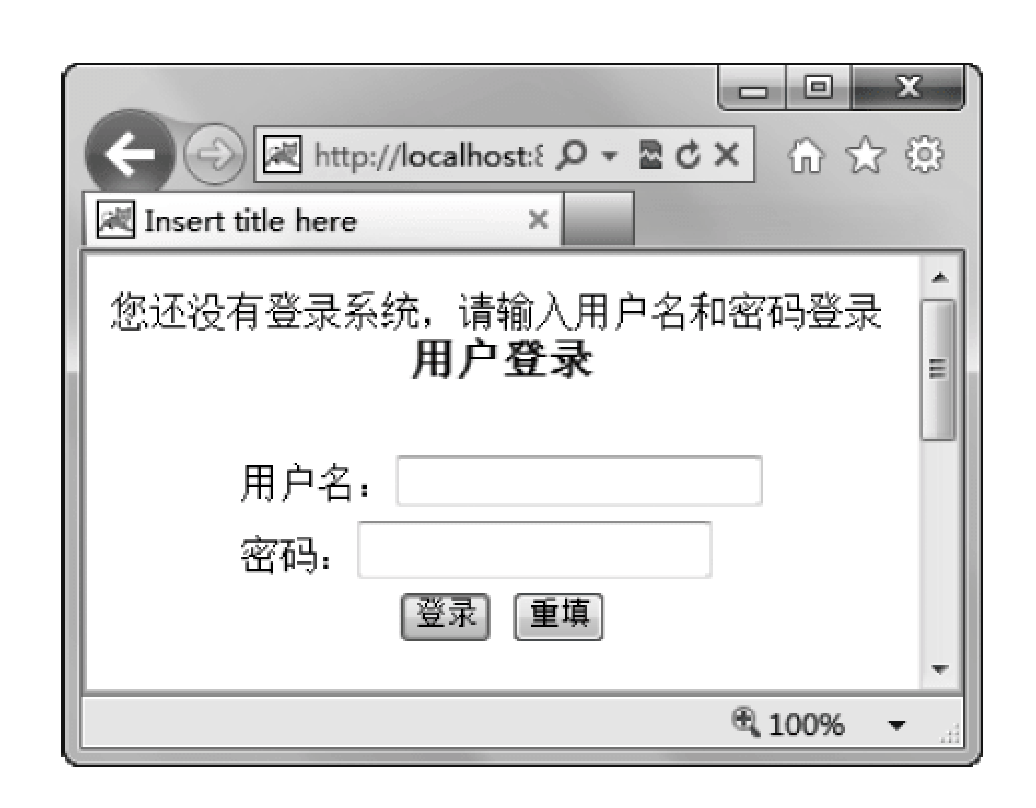


图 19.17 login.jsp 权限验证提示信息

19.6.2 拓展任务 4:完成用户管理模块拦截器应用

任务要求:完成论坛管理系统用户管理模块拦截器的应用,包括数据库访问异常配置、防止表单重复提交配置等。

19.6.3 实训 4:实现论坛管理系统的主题管理模块的拦截器应用

实训要求:完成论坛管理系统主题管理模块拦截器的应用,包括权限验证拦截器、数据库访问异常配置、防表单重复提交配置等。

19.7 本章小结

本章以论坛管理系统的用户管理模块为例,讲解了使用 Struts 2 框架开发系统的过程,包括以下 6 步。

(1) 系统分析与设计

① 需求分析

② 功能设计

③ 数据库设计

(2) 制作命名规范

(3) 搭建系统架构

① 创建 Struts 2 应用

② 创建视图层 JSP 文件
③ 创建 Action 类
④ 配置 Action
⑤ 编写视图层文件请求
（4）实现 Struts 2 框架中 MVC 各层
① 创建模型类
② 实现业务逻辑组件
③ 实现 Action 类功能
④ 实现视图层功能
（5）实现 Struts 2 框架的数据输入校验
（6）配置 Struts 2 拦截器

通过较完整地开发系统的一个模块，我们掌握了应用 Struts 2 框架开发系统的步骤，熟练了技能的应用，并锻炼了综合应用技术的能力。

本章提示了在哪些步骤进行测试，可以提高程序的调试效率，保证系统的顺利开发。

本章还讲解了使用 Struts 2 框架开发系统的方法和技巧，包括如何设计命名规范，如何综合应用技术开发系统，以及如何包含配置文件以方便开发组成员分工合作等。

通过本章的学习，读者已经可以独立开发应用 Struts 2 框架的系统。

参 考 文 献

[1] 李明革,孙佳帝. Java Web应用教程——网上购物管理系统的实现[M]. 北京：中国人民大学出版社,2011.
[2] 刘晓华,张健,周慧贞. JSP应用开发详解[M]. 北京：电子工业出版社,2007.
[3] 李刚. Struts 2. x权威指南[M]. 北京：电子工业出版社,2012.
[4] 蒲子明. Struts 2＋Hibernate＋Spring整合开发技术详解[M]. 北京：清华大学出版社,2010.